Thèse

présentée pour obtenir le grade de docteur

de l'Ecole nationale supérieure des télécommunications

Spécialité : Electronique et Communications

Luis Garcés-Erice

Un Réseau P2P Hiérarchique: Design et Applications

soutenue le 6 Décembre 2004 devant le jury composé de

Prof. Isabelle Demeure	**Président**
Dr. Anne-Marie Kermarrec	**Rapporteurs**
Prof. Marcel Waldvogel	
Prof. Ernst W. Biersack	**Directeur de thèse**

Ecole nationale supérieure des télécommunications

A todos los que han contribuido a este trabajo, bien con su apoyo, con su sabiduría, su amistad o su cariño.

À tous ceux qui ont contribué à ce travail, soit avec son support, sa sagesse, son amitié ou son amour.

To all those that have contributed to this work with either their support, their wisdom, their friendship or their love.

Table des matières

Table des figures

Liste des tableaux

Résumé

Le succès des connexions Internet à haut débit et la baisse des prix des systèmes de calcul comme les ordinateurs personnels ont contribué à favoriser l'essor de systèmes qui ne dépendent plus de l'infrastructure centralisé d'une tierce partie. Ces systèmes qui sont surgis aux bords de l'Internet sont nommés systèmes Pair-à-pair (*Peer-to-peer*, P2P).

Deux grandes familles de systèmes P2P peuvent être considérées : structurées ou non-structurées. La différence principale entre les deux repose sur l'algorithme de recherche utilisé pour trouver des ressources dans le réseau P2P.

Les systèmes P2P structurés possèdent un algorithme de recherche pour lequel une ressource donnée est assignée de façon univoque à un pair, (pour un état donné du système). L'assignation est faite en minimisant une métrique prédéfinie sur un espace numérique d'identificateurs partagé par les pairs et les ressources. L'algorithme de recherche est complètement déterministe, et les liens entre les pairs sont faits suivant des règles bien définies. L'infrastructure de recherche marche comme un Tableau de Hachage Distribué ou THD (*Distributed Hash Table*, DHT) implémenté à travers les pairs.

Les réseaux P2P non-structurés ne font pas correspondre des pairs et des ressources de façon stricte, du à l'absence d'un espace commun d'identificateurs, ou tout simplement à l'absence des identificateurs au sens propre. L'algorithme de recherche est aléatoire dans le sens où la recherche d'une donnée n'a pas un but préétabli qui puisse définir sa finalisation. Les connexions entre les pairs sont aussi aléatoires, parce qu'elles sont établies par convenance ou sans aucune règle, n'ayant pas *a priori* de relations prédéfinies ou nécessaires entre les pairs.

Dans cette thèse nous nous concentrons sur les systèmes P2P structurés, séduit par sa nouveauté et le passage à l'échelle des propriétés des THDs. Nous étudions en particulier quelques aspects de la Distribution et Localisation de Contenus, des sujets capitaux dans les services traditionnels de l'Internet (spécifiquement, son application majeure, le WWW) et que nous les traduisons sur des réseaux P2P structurés.

Dans notre étude, nous avons suivi une approximation de bas en haut. Nous commençons par considérer l'architecture de THDs, en présentant un cadre général pour des systèmes P2P hiérarchiques, sur lequel on développe après les autres sujets. Nous établissons les principales caractéristiques et avantages de l'organisation hiérarchique : plus spécifiquement, la capacité de ce type des systèmes à améliorer la communication réseau en s'adaptant à la topologie de l'Internet, et à incrémenter la robustesse des THDs traditionnelles par la différenciation des rôles

des pairs selon ses capacités.

Prenant le cas particulier des systèmes P2P hiérarchiques étudiés, nous avons développé un THD nommé TOPLUS, construit par le groupage des pairs avec le même préfixe de réseau IP, et l'organisation de ces groupes en suivant la topologie hiérarchique de l'Internet. Nous présentons les caractéristiques de TOPLUS, et nous étudions de façon plus détaillée son but principal : Un algorithme de recherche capable de trouver le pair responsable d'une ressource dans un temps comparable a celui du routage du niveau-3 de la requête depuis le pair à l'origine de la même jusqu'au pair de destination. Parce que la structure de TOPLUS suit celle de l'Internet, on dit que TOPLUS est *topology-aware* (tient compte de la topologie).

Ensuite, nous proposons des solutions pour la Distribution et Localisation des Contenus sur des systèmes P2P structurés.

Pour la distribution des contenus on présent MULTI+, un réseau P2P pour multicast applicatif qui se place à un niveau au dessus de TOPLUS. MULTI+ construit des arbres multicast avec le plus court délai possible de la source aux récipients dans le groupe multicast, en essayant d'utiliser la bande passante de façon efficace. Ces buts sont atteints au moyen d'algorithmes de construction d'arbres multicast qui profitent de la adaptation à la topologie faite par le THD TOPLUS. Nous vérifions aussi combien les arbres MULTI+ sont robustes au niveau de la préservation des objectifs de délai et bande passante en présence d'erreurs ou abandon massif des pairs.

Finalement, nous introduisons une technique de Localisation de Contenus pour des systèmes P2P structurés : nous utilisons pour cela l'indexation des ressources avec des schémas d'indexation prédéfinis. Chaque index pointe vers un ou plusieurs index qui contiennent une information plus précise sur une ressource donnée, et l'index qui décrit complètement une ressource pointe directement sur ladite ressource. Le pair dans le réseau P2P trouvée par l'algorithme de recherche responsable de l'identificateur d'un index stocke tous les index pointés par le premier. De cette manière, l'infrastructure de localisation de contenus est incluse dans le système P2P et hérite de toutes ses propriétés.

Liste des abréviations

Pour des raisons de lisibilité, la signification d'une abréviation ou d'un acronyme n'est souvent rappelée qu'à sa première apparition dans le texte d'un chapitre. Par ailleurs, puisque nous utilisons toujours l'abréviation la plus usuelle, il est fréquent que ce soit le terme anglais qui soit employé, auquel cas nous présentons une traduction.

AS	Autonomous System		
BGP	Border Gateway Protocol		
CCDF	Complementary Cumulative Distribution Function		
CDF	Cumulative Distribution Function		
DHT	Distributed Hash Table	**THD**	Tableau de Hachage Distribué
DNS	Domain Name System		
FIFO	First In, First Out		
ID	Identifier		Identifieur
IP	Internet Protocol		
ISP	Internet Service Provider		Fourniseur d'Accès Internet
LAN	Local Area Network		Réseau Local
LRU	Least Recently Used		
MSD	Most Specific Descriptor		
P2P	Peer-to-Peer		Pair-à-Pair
RIG	Responsible Inner Group	**GIR**	Groupe Interior Responsable
RTT	Round Trip Time		
TCP	Transport Control Protocol		
WAN	Wide Area Network		
XML	eXtended Mark-up Language		
XOR	eXclusive OR (bit-wise operation)		
XPath	an XML-based query construction language		Langage basé sur XML pour la construction de requêtes

Liste des notations

Nous avons regroupé ci-dessous les principales notations employées dans les différents chapitres du document. Dans la mesure du possible, nous avons tenté de conserver les mêmes notations d'un chapitre à l'autre. Nous présentons tout d'abord une liste générale puis des listes relatives aux différents chapitres. On notera que seules les notations qui diffèrent de celles précédemment définies seront données dans ces listes. Enfin, certaines notations, apparaissant uniquement de manière ponctuelle, ont été omises.

Notations générales

k	Une clé.
$p, q, r, \ldots$	Chaque lettre est un pair. En abusant de la notation, chaque lettre est aussi le ID d'un pair.
$p, p', p'', \ldots$	Des pairs concernés par une opération P2P.
$G, G', \ldots$	Chaque lettre est un groupe des pairs. En abusant de la notation, chaque lettre est aussi le ID d'un groupe de pairs.
$\|G\|$	Nombre des pairs dans un groupe G.
N	Nombre des pairs dans un système P2P.
U	Ensemble des pairs disponibles à un moment donné dans un système P2P.
$d(i, j)$	Pour deux ID i et j, la distance entre eux selon une certaine métrique (e.g., la métrique XOR).
$d(p, q)$	Pour deux pairs p et q, la distance entre eux selon un certaine métrique (e.g., latence).
$d(G, G')$	Pour deux groupes G et G', elle est égale à la distance $d(p, q)$ où $p \in G$ et $q \in G'$.

Chapitre 2

G_i	ID du groupe i. En abusant la notation, l'ensemble de pairs dans le groupe i.
p_i	Un pair appartenant au groupe i. En abusant la notation, l'ID du pair.
S_i	L'ensemble des *super* pairs dans le groupe G_i.
s_i	Un superpair du groupe G_i. En abusant la notation, l'ID du pair.
R_i	L'ensemble des pairs *normaux* dans le groupe G_i. $R_i = G_i - S_i$
r_i	Un pair normal dans le groupe G_i. En abusant la notation, l'ID du pair.

I	Le groupe contenant tout les groupes d'un arbre TOPLUS.
L	Nombre maximal de couches dans un arbre TOPLUS.
$H_i(p), \cdots$	Ensemble des groupes contenants un pair p dans un arbre TOPLUS.
$\cdots, H_0(p) = I$	
$\mathcal{S}_i(p)$	Ensemble des groupes jumeaux de $H_i(p)$.
$\mathcal{S}(p)$	$\bigcup \mathcal{S}_i(p)$.
$d^T(G, G')$	La distance entre les groupes G et G' a travers du chemin TOPLUS.
$d^{IP}(G, G')$	Égale à $d(G, G')$, utilisant comme métrique la latence IP.

Chapitre 3

m	Une adresse multicast ; un groupe multicast. La clé qui identifie un groupe multicast.
m_i	La clé résultante de la substitution de ses premièrs bits par ceux du préfixe du réseau IP d'un groupe dans la couche-i d'un arbre TOPLUS.
GIR-i	*G*roup *I*ntérieur *R*esponsable dans une couche-i d'un arbre TOPLUS. C'est le groupe responsable de la clé m_i.
$d, d_1, d_2, \ldots$	Descripteurs de ressources.
$\mathsf{q}, \mathsf{q}_1, \mathsf{q}_2, \ldots$	Requêtes pour des ressources (expressions XPath).

Chapitre 1

Introduction

1.1 L'arrivée des Systèmes Pair-à-Pair

L'Internet que l'on connaît aujourd'hui aurait pu devenir beaucoup de choses depuis les jours lointains de l'Arpanet. Le réseau qui un jour permettra de communiquer entre deux points quelconques de la planète aurait pu être aujourd'hui une infrastructure intelligente, prête à combler les besoins de ses millions d'utilisateurs [1]. Offrant une myriade de services, le réseau pourrait s'adapter aux comportements et préférences des gens pour rendre son utilisation transparente. Ouvrant les portes de ce paradis de la communication aux masses, les fournisseurs d'accès pourraient offrir des contenues exclusifs à ses utilisateurs au travers de services à la demande ou en échange d'un abonnement mensuel pour que la connexion à Internet en vaille la peine [2]. Tout ça aurait pu arriver, mais ce ne fut pas le cas. En effet, l'Internet n'est pas la télé par câble au milliard de chaînes différentes ; l'Internet n'a pas un véritable point central de contrôle, et ceux qui offrent des informations utilisent la même technologie que ceux qui la reçoivent.

L'Internet aujourd'hui est, à la base, l'infrastructure physique et la collection de protocoles (tels que 802.3 Ethernet ou la famille des protocoles dits "sans-fil" 802.11 aux bords de l'Internet, et ATM ou *Frame Relay* au coeur) dont IP [3] a besoin pour envoyer des informations d'un hôte à un autre. Le Protocole Internet fut conçu au début comme un moyen pour connecter les différentes réseaux existants (qui utilisaient des protocoles comme SNA d'IBM ou DECNET de Digital [4]), pour créer un *interéseau* (Internet). Le but de cet interéseau était de permettre la communication entre les participants, et ainsi l'échange de données entre ses réseaux devint naturellement la *fonction la plus importante*. Le protocole qui assurait cet importante fonction, IP, devint à son tour le protocole réseau omniprésent, en transférant simplement [5] des paquets d'un bout a l'autre de l'Internet [6].

Toute communication nécessite au moins deux parties. L'Internet, comme tout autre moyen de communication, n'a aucun intérêt en soit. Cela vaut la peine de posséder une maison, ou

de payer pour une voiture : un individu peut obtenir un bénéfice direct de ces biens sans la participation des autres. Ce n'est pas le cas des moyens de communication. Les usagers perçoivent de la valeur en eux parce que d'autres utilisateurs participent pour envoyer et recevoir des données [8]. Pourtant, les moyens nécessaires pour l'échange des données, c'est-à-dire, les machines et les lignes de communication, étaient trop chères par le passé. Les utilisateurs individuels dépendaient des fournisseurs des services sur Internet (ISP, Internet Service Provider) pour recevoir, envoyer (e-Mail [9], News Forums [10]), ou publier (pages Web [11], serveurs FTP [12]) de l'information.

La plus grande généralisation de l'utilisation d'Internet [13, 14, 8] créât des opportunités pour des tierces parties, lesquelles commençaient à offrir de services sans fournir l'accès eux mêmes. Le service E-mail à travers une interface Web avec du stockage distant, connu sous le nom de web-mail, et les services de messagerie instantanée de personne à personne (IM, Instant Messaging [15]), ainsi que le commerce électronique (voir [16] et aussi [4], Section III.A) sont des bons exemples de ces initiatives. Or, les usagers restaient encore dépendants de l'infrastructure fournie par ces tierces parties, principalement à cause des ressources requises pour offrir ces services à un grand nombre d'utilisateurs de façon efficace.

Au cours des années, avec l'arrivée du PC, les masses étaient en mesure d'acheter de machines puissantes [17]. La concurrence entre ISPs pour capter des nouveaux clients fit baisser les prix des accès à Internet, en même temps qu'ils augmentaient la bande passante offerte aux utilisateurs finales. N'importe qui ayant un PC et une connexion Internet avait assez des ressources pour fournir les services qui par tradition correspondaient aux ISPs, comme l'e-Mail ou l'hébergement de pages web, une possibilité en partie favorisée par l'apparition des logiciels du domaine public pour les systèmes d'exploitation (e.g., distributions de software basés sur GNU/Linux) et des applications (e.g., le serveur Web Apache). Bien que le saut entre une situation où les gens payent pour une adresse e-Mail et une situation où ces mêmes gens peuvent offrir des adresses e-Mail à ses amis dans son propre domaine soit impressionnante, en réalité, très peu nombreux sont les utilisateurs offrant les services Internet traditionnels (hébergement des pages Web, serveur FTP, etc).. Même quand c'est le cas, les ressources offertes sont limitées, ce qui restreint ces services aux petites communautés.

Ainsi, même en ayant toutes ces ressources aux bords de l'Internet, apparemment les "vieux" services ne pouvaient pas bénéficier de ce développement, et pour une bonne raison : ces services n'étaient pas conçus pour une telle situation. Les utilisateurs étaient censés *utiliser* les servies, et non les *offrir*, et ainsi les PCs de millions d'usagers d'Internet demeuraient passifs et inactifs la plupart du temps. Ils leur manquaient la technologie pour réunir les ressources éparpillées sur les bords de l'Internet. Et c'est à ce moment que les systèmes (Gnutella [18], Fast-Track) et applications (Napster [19], KaZaA [20], E-Mule [21], Grokster, LimeWire, Morpheus, etc). P2P ont fait leur apparition.

1.2 Systèmes Pair-à-Pair

Les systèmes P2P s'appuient sur beaucoup des concepts du domaine des systèmes distribués : Algorithmes pour la redondance, équilibrage de la charge, réseau *overlay* ou la robustesse sont des briques de base de ces systèmes. Mais, dans quel sens son les systèmes P2P sont-ils nouveaux ? Qu'est-ce qui fait leur spécificité ?

Des systèmes avec un comportement Pair-à-Pair existaient bien avant les applications dites "P2P" d'aujourd'hui. Aux balbutiements de l'Internet, tous les ordinateurs étaient des *pairs*, parce qu'ils n'y avaient pas un système DNS (Domain Name System) en place, et une machine devait connaître une autre à l'avance pour pouvoir établir une communication avec elle. Alors, pourquoi est-ce qu'on parle autant [24] du P2P ces jours-ci ? Parce que le genre de systèmes dont il est question ne pouvaient même pas être imaginés à l'époque de cet Internet primitif "Pair-à-Pair". C'est seulement aujourd'hui que l'on peut commencer à construire des systèmes constitués par de millions d'hôtes interconnectés. Les systèmes P2P sont la conséquence du développement de l'Internet. Nous proposons notre propre définition, comme suit :

Les systèmes P2P son des applications distribuées et massivement déployées, supportées par les ressources fournis pas les usagers aux bouts de, et interconnectés par, l'Internet.

C'est seulement aujourd'hui que ces systèmes sont faisables. En tant que tels, les systèmes Pair-à-Pair présentent leurs propres caractéristiques :

- Ils sont composés d'un grand nombre d'hôtes, et donc leurs propriétés doivent passer à l'échelle pour un nombre grandissant de participants. Des millions d'usagers peut être considéré comme une situation tout à fait normale [25, 26, 27].
- La capacité de calcul est fournie par les machines des utilisateurs, et ces machines peuvent être puissantes mais en tout cas pas fiables. Des composantes standards doivent être supposés dans toutes les machines. N'importe quelle machine (pair) peut à tout moment tomber en panne [28] ou se déconnecter [26].
- On peut justifier une certaine bande passante requise pour le fonctionnement correct de certaines applications P2P. Pourtant, en général, la bande passante doit être considérée comme une ressource peu abondante [25, 26]. Donc un sens de l'économie doit être associé à toute communication entre pairs qui ne concerne pas les échanges d'informations entre les usagers eux mêmes.
- Les usagers sont intelligents. Ils savent qu'ils doivent collaborer pour obtenir un bénéfice du système. Pourtant, la plus part des usagers n'offriront pas ses ressources sans un intérêt qui les motive. Dans la vie réelle, dans les réseaux P2P, très peu de pairs offrent de ressources de façon désintéressée, pendant que certains ne collaborent pas du tout. Au milieu se trouvent les utilisateurs normaux, qui construisent l'infrastructure commune motivés par leur propre intérêt. Un exemple de comportement *avare*[1] dans les réseaux de partage

[1] On utilise le terme *avare* pour souligner le fait que les pairs trouvent important que les autres n'utilisent pas ses ressources. Un pair *égoïste* collabore, motivé par son propre intérêt, parce que la collaboration de tous les pairs est nécessaire pour le bon fonctionnement du réseau.

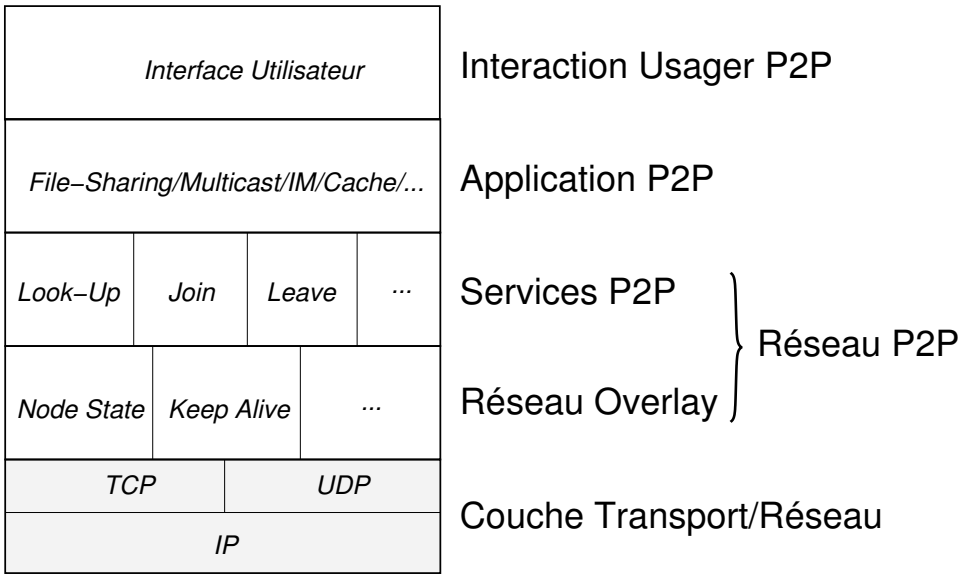


FIG. 1.1 – La pile des protocoles P2P.

de fichiers sont les *freeriders* [29].
Les freeriders ne partagent pas de fichiers, mais ils téléchargent des contenus depuis les autres pairs. D'autres pairs avares font partie du système P2P seulement le temps de satisfaire leurs intérêts, et ils se déconnectent tout de suite après. Les pairs avares, pendant qu'ils sont connectés se comportent comme des pairs normaux, seulement ils n'ont pas de ressources à offrir. Les pairs qui se connectent au réseau P2P et refusent de suivre les protocoles (ou qui font carrément du sabotage) sont un risque pour la sécurité. Les pairs avares ne représentent pas un tel danger. Bien au contraire, ils sont un exemple des différences entre les systèmes P2P et les systèmes distribués traditionnels : ces derniers sont des infrastructures dédiés, mais les premières sont rationnelles. Ceci ne peut pas être ignoré, et les systèmes P2P peuvent implémenter des mécanismes pour forcer les pairs à collaborer. Dans le pire de cas, un système P2P composé uniquement de pairs avares cesse d'exister parce qu'aucun pair ne peut obtenir quelque chose des autres.

En général, un système P2P est (plus ou moins) composé d'un substrat permettant la communication entre les pairs, les algorithmes trouvant les ressources et une application au dessus de l'environnement distribué. On passe maintenant aux définitions des éléments principaux d'un réseau P2P.

Sur la Figure 1.1 on peut voir une pile détaillée des protocoles P2P. Remarquez que toutes les couches P2P sont situées dans la *couche application* du modèle OSI.

Ce que l'on appelle *Réseau P2P* est composé de deux couches très couplées : le *Réseau Overlay* et les *Services P2P*.

– *Réseau Overlay* : Une méthode de routage de niveau applicatif entre les noeuds d'une application distribuée. Les paquets voyagent d'un noeud à l'autre routés par la couche réseau (OSI niveau 3), mais la *décision de routage* (i.e., quel noeud est la destination à

chaque saut) est faite explicitement au niveau applicatif à chaque noeud final. C'est la raison pour laquelle les réseaux overlays sont aussi appelés Réseau Applicatifs. Un exemple solide est le *Resilient Overlay Network* (RON, Réseau Overlay Robuste) : Un groupe de noeuds multi-home assurent la continuité de la communication même en présence des pannes du réseau, en utilisant les chemins sur l'overlay de niveau applicatif pour éviter les réseaux ou les liens en panne. Les réseaux overlay sont représentés d'habitude par un graphe dirigé (X,U), ou X est l'ensemble des noeuds terminaux dans l'overlay et U est l'ensemble des liens de niveau applicatif entre les noeuds dans X. Le graphe (X,U) doit être connexe.

– *Services P2P* : La fonctionnalité fournie par le réseau P2P pour les application au dessus. Chaque instance d'une application distribuée dans chaque pair communique avec les autres instances sur d'autres pairs à travers l'API offerte par les services P2P. Des exemples de ces fonctions dans l'API sont la connexion à un réseau P2P (join), ou la recherche (look-up) du pair responsable d'une certain ressource dans le réseau P2P.

Dorénavant, nous utiliserons le terme *réseau P2P* en référence à tout un système Pair-à-Pair, jusqu'à la application P2P. Même si nous ne sommes pas d'accord avec cet identification d'un système par l'une de ses parties, cet terminologie est fréquemment utilisée et très répandue. Nous croyons que cette ambiguïté ne trompera pas le lecteur, car le contexte pourra clarifier la portée du terme (système complet ou seulement le réseau P2P). Quand l'on utilise le terme *réseau* seul, on fait référence au réseau physique en dessous jusqu'à la couche 3 du modèle OSI, dans notre cas, IP.

1.3 L'Algorithme de Recherche dans un Système P2P

Les ressources dans un système P2P sont distribuées entre tous les pairs. Ces ressources doivent d'abord être trouvées (et donc cherchées) pour pouvoir y accéder. Une *clé* identifie un ressource dans un système P2P. La forme la plus générale qu'on peut trouver dans la bibliographie est une chaîne de caractères ou une valeur numérique de taille fixe. Selon comment un système P2P cherche les ressources en utilisant ces clés, et selon la relation entre les clés et les pairs, le type de système P2P est défini différemment, comme on va le voir tout de suite.

Les systèmes décrits dans la section précédente peuvent être encore divisé entre systèmes structurés et non-structurés. La différence principale entre les deux repose sur la méthode employée pour chercher les ressources dans le réseau P2P.

Nous définissons l'Algorithme de Recherche comme l'algorithme distribué de niveau applicatif implémenté dans chaque noeud d'un réseau overlay, de façon à permettre que une requête pour la clé k arrive au noeud responsable de la dite clé.

Les réseaux overlay et de niveau 3 routent des *paquets*. Dorénavant, nous appelons les données routées par les réseaux P2P des *requêtes*. Dans les deux types de systèmes P2P, structurés et non-structurés, chaque pair connaît un nombre réduit (comparé à la population totale du ré-

seau P2P) de pairs appelés *voisins*. La population des pairs et les connections vers ses voisins forment le graphe (X,U) qui est le substrat du réseau P2P. Mais les voisins d'un pair (et donc les liens en U) sont choisis selon l'algorithme de recherche utilisé dans le réseau P2P. Donc on peut dire que l'algorithme de recherche défini un réseau P2P, parce qu'il détermine le graphe (X,U) du réseau overlay (voir la Figure 1.1). L'ensemble des voisins d'un pair est appelé *tableau de routage*, parce que le prochain saut pour une requête est décidé parmi les voisins du pair qui prend la décision de routage. Un papier fort intéressant [32], qui étudie la relation entre le diamètre du réseau et la taille du tableau de routage, montre l'intime équilibre entre l'algorithme de recherche et les voisins dans un réseau overlay.

Nous passons à l'exploration des différences entre les systèmes P2P structurés et non-structurés, concernant les clés, les pairs et l'algorithme de recherche. Nous allons utiliser pour cela deux exemples bien connus : Gnutella [18] et Chord [33].

Systèmes P2P Non-structurés :

- *Clés :* Elles sont typiquement une chaîne de caractères, appartenant à un espace de noms de taille arbitraire. D'habitude, les clés contiennent la description de la ressource qu'elles identifient.
  Exemple : dans des applications de partage des fichiers comme Gnutella, chaque clé est le nom d'un fichier.
- *Pairs :* Les pairs ne sont identifiés par un nom propre à eux. Les pairs hébergent des ressources pour lesquels l'utilisateur a un intérêt explicite. Les voisins d'un pair sont choisis au hasard parmi le nombre total des pairs, même si des optimisations peuvent être introduites pour améliorer l'algorithme de recherche, comme se connecter vers des pairs proches, ou vers ceux qui offrent l'information la plus relevant. Ce dernier critère conduit à la formation de réseau *Power-law* [35], ou la plupart des pairs sont connectés à un ensemble réduit des pairs. On dit que ces pairs ont un haut *in-degree*, c'est-à-dire, beaucoup de liens dans (X,U) sont dirigés vers ces pairs là. Inversement, on dit que ces quelques pairs ont un haut *fan-out*, c'est-à-dire, qu'ils servent des contenus à beaucoup d'autres pairs.
  Exemple : Dans [35], les auteurs suggèrent la possibilité de choisir comme voisins les pairs avec le plus grand nombre de connections vers d'autres pairs, pour améliorer l'efficacité de la recherche. Le phénomène du "petit monde" (*Small World*) [36] montre combien la recherche des données dans ce type de réseau peut être efficace.
- *Algorithme de Recherche :* Quand un pair cherche une ressource donnée, il envoie une requête qui contient des informations qui doivent correspondre à la clé identifiant le ressource. La requête est envoyée à un certain nombre (peut-être tous) de voisins, lesquels cherchent la clé correspondante parmi les ressources qu'ils stockent. Ces pairs renvoient à leur tour la requête originelle à ses propres voisins, pour que la recherche atteigne tous les pairs. Les clés correspondantes sont renvoyées au pair à l'origine de la requête comme résultat de la recherche.
  Exemple : Sur Gnutella, un pair qui reçoit une requête cherche parmi les fichiers qu'il stocke renvoi une liste des noms de fichiers correspondants au pairs d'où vient la requête et ré-émet la requête à ses propres voisins. Le processus [37] finit après un certain nombre

(faible) de saut, pour éviter d'inonder le réseau de messages. KaZaA fonctionne de façon similaire, mais il introduit un type spécial de pairs, les superpairs (*superpeers*), qui connaissent les contenus stockés chez d'autres pairs et qui peuvent répondre aux requêtes à sa place.

Systèmes P2P Structurés :

- *Clés :* Dans les systèmes P2P structurés, les clés sont des chaînes de bits de taille fixe. c'est-à-dire, des identificateurs numériques (ID) d'un ensemble fini. Il n'y a pas une relation explicite entre un ressource et la clé qui l'identifie. Une méthode directe pour l'obtention des clés à partir des ressources est le hachage : le résultat de l'application d'une fonction de hachage à une ressource ou des méta-données référant à la dite ressource est la clé. Ceci introduit une relation implicite entre la clé et la ressource.
  Exemple : Dans Chord, une clé est une chaîne de 160 bits. Les clés sont obtenues à partir d'une fonction de hachage sécurisée SHA-1 [38].
- *Pairs :* Les pairs sont identifiés de façon unique par un ID numérique obtenu du même ensemble que les clés. On abuse de la notation, dénotant un pair et son ID par le même symbole, généralement p. Les voisins d'un pair son définis de façon précise par l'algorithme de recherche. Un pair est responsable des ressources lui correspondant selon l'algorithme de recherche. Un pair p est responsable d'une clé k si p est le pair avec l'ID le plus proche de k dans le réseau P2P, selon une métrique prédéfinie.
  Exemple : Chez Chord, pour un espace d'identificateurs I des chaînes de m bits, un pair d'ID $p \in I$ possède un ensemble de voisins défini par les pairs avec ID $q_i = (p + 2^i) \mod 2^m, 0 \leq i < m$. Un pair p est responsable de toutes les clés k telles que $q < k \leq p$, ou q est le pair avec le plus grand ID plus petite que p.
- *Algorithme de Recherche :* Quand un pair p envoie une requête pour une clé k, p envoie la requête au pair voisin avec l'ID le plus proche de k, selon une certaine métrique. Chaque pair recevant la requête répète la même opération, jusqu'à que finalement le pair responsable de k soit atteint par la requête.
  Exemple : Chaque pair p sur Chord renvoi une requête pour une clé k au pair q dans le tableau de routage de p avec le plus grand ID, mais respectant $k < q$.

Les systèmes P2P structurés sont souvent identifiés aux Tableaux de Hachage Distribués (*Distributed Hash Table, DHT*), ou THD. En fait, c'est plutôt l'algorithme de recherche des systèmes P2P structurés qui est identifié à cette structure de données. En effet, un Tableau de Hachage (*Hash Table*) [39] donne l'entrée dans un tableau correspondant à une clé. Dans ce sens là, l'algorithme de recherche P2P fait à peu près la même chose, il donne le pair correspondant à une clé. Ainsi, nous définissons :

Un Tableau de Hachage Distribué (THD) est un algorithme de recherche dans un système P2P structuré qui fait correspondre de façon unique une clé k à un pair p, et qui définit le substrat d'un réseau overlay (X,U) au travers duquel les requêtes pour k sont routées vers p.

Les algorithmes de recherche offrants une fonctionnalité THD se différencient les uns des autres dans la manière dont les requêtes pour les clés sont routées pour trouver les pairs responsables. Nous considérons l'identification des systèmes P2P structurés avec des THDs valide

quand il est clair que le système P2P ne fournie qu'une fonctionnalité de recherche (c'est-à-dire, quand nous considérons que le système P2P n'est formé que par le réseau P2P uniquement). Dans le cas d'un système P2P plus général, on préfère le terme de *Service de Recherche P2P* (*P2P Look-Up Service* (comme dans la Figure 1.1) pour nommer l'implémentation du THD utilisé comme algorithme de recherche. Le système correspondant peut être nommé Système P2P basé sur THD (*DHT-based P2P system*).

Les systèmes P2P non-structurés ont été présentés ici pour compléter la vision du domaine des systèmes P2P, mais ils ne seront pas l'objet d'étude ultérieur dans cet thèse. On y fera référence si besoin ait pour clarifier ou comme analogie.

À partir de maintenant nous focalisons notre travail sur les systèmes P2P structurés. Des exemples de tels systèmes sont Chord [33], CAN [40], Pastry [41], Tapestry [42], et P-Grid [43].

Chapitre 2

THD Hiérarchique : Un Design

2.1 Réseaux P2P Hiérarchiques

Chord, CAN, Pastry et Tapestry sont tous des THDs à design horizontal sans routage hiérarchique. Tous les pairs sont identiques dans le sens où tout les pairs utilisent les mêmes règles pour déterminer le routage d'une requête. Cette approche est très différente de celle de l'Internet, où le routage hiérarchique est utilisé. En effet, dans l'Internet les routeurs sont groupés dans des *Autonomous Systems* (AS). Dans un AS, tous les routeurs utilisent le même protocole de routage intra-AS (e.g., RIP ou OSPF). Certains routeurs qui servent de passerelles entre les AS utilisent des protocoles de routage inter-AS (e.g., BGP) chargés de déterminer le chemin de routage d'AS à AS. Le routage hiérarchique dans l'Internet offre d'importants avantages par rapport au routage horizontal, telles que le passage à l'échelle ou l'autonomie d'administration (e.g., au niveau d'un campus universitaire, d'une entreprise, ou de la zone de couverture d'une station de base pour un réseau mobile).

Dans ce chapitre on explore les THDs hiérarchiques. Inspirés par le routage hiérarchique sur Internet, nous examinons les THDs à deux couches dans lesquelles

1. les pairs sont organisés dans de groupes différents, et
2. les messages de recherche sont d'abord routés vers le groupe de destination en utilisant un overlay entre les groupes, et après vers le pair de destination en utilisant un overlay à l'intérieur du groupe.

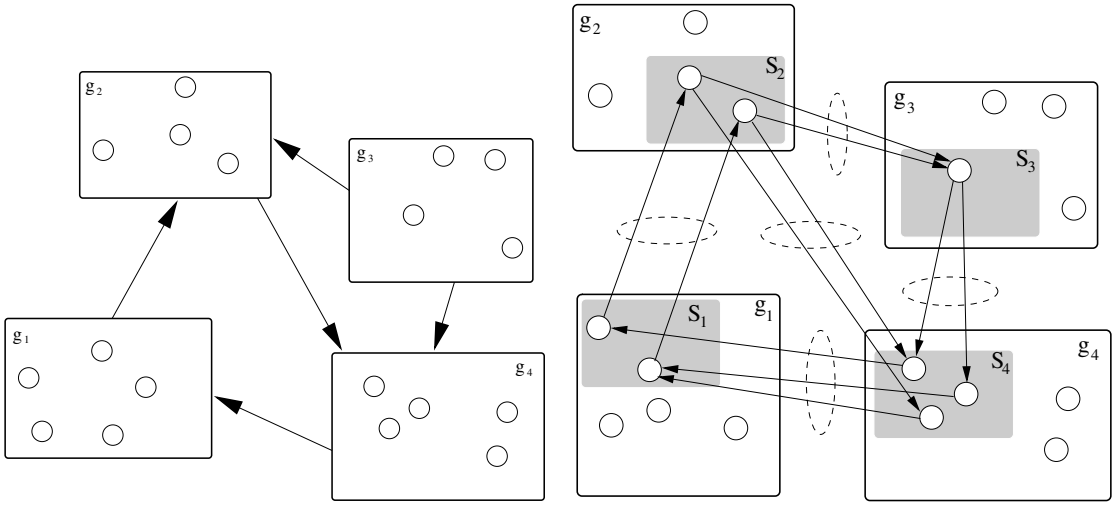
Soit N l'ensemble des pairs dans le système. Chaque pair a un ID. Chaque pair a aussi une adresse IP, qui peut changer à chaque fois que le pair se reconnecte au système. Les pairs sont interconnectés par un réseau de liens et d'équipement de commutation (routeurs, etc).. Les pairs envoient des messages de recherche les uns vers les autres utilisant le réseau overlay hiérarchique.

Les pairs sont organisés en groupes. Les groupes peuvent être composés de pairs qui sont

proches les uns des autres, si l'application nécessite cette propriété. Chaque groupe a un ID unique. Soit I le nombre de groupes, G_i les pairs dans un groupe i, et, par abus de la notation, l'ID du groupe i.

Les groupes sont organisés en un *réseau overlay de niveau supérieur* défini par un graphe dirigé (X,U), ou $X = \{G_1,\ldots,G_I\}$ est l'ensemble des groupes et U est l'ensemble des liens virtuels entre les noeuds (groupes) dans X. Le graphe (X,U) doit être connexe, c'est-à-dire, entre deux noeuds G et G' dans X il y a un chemin dirigé depuis G jusqu'à G' qui utilise les liens dans U. Il est important de remarquer que ce réseau overlay défini des liens dirigés entre les groupes et pas entre des pairs spécifiques dans chaque groupe.

Chaque groupe doit avoir un ou plusieurs *superpairs*. Les superpairs ont des caractéristiques et des responsabilités spéciales. Soit $S_i \subseteq G_i$ l'ensemble des superpairs du groupe i. Notre architecture permet d'avoir $S_i = G_i$ pour tous les $i = 1,\ldots,I$, et dans ce cas tous les pairs sont des superpairs. Les architectures pour lesquelles tous les pairs sont des superpairs sont dites à *design symétrique*. Notre architecture permet aussi $|S_i| = 1$ pour tous les $i = 1,\ldots,I$, et dans ce cas chaque groupe a un seul superpair. Soit $R_i = G_i - S_i$ l'ensemble de tous les "pairs normaux" du groupe G_i. Pour les designs non-symétriques ($S_i \neq G_i$), il faut essayer de désigner les pairs les plus puissants comme superpairs. Par "plus puissants" nous voulons dire les pairs qui restent connectés le plus longtemps. Mais comme critère secondaire, les superpairs peuvent être aussi ceux qui ont une bande passante importante ou une grande capacité de calcul.



(a) Entre groupes. (b) Entre les superpairs des groupes voisins.

FIG. 2.1 – Relations de communications dans le réseau overlay.

Les superpairs sont des passerelles entre les groupes : ils sont utilisés pour la propagation des requêtes entre les groupes. À cette fin, on demande que si s_i est un superpair dans G_i, et (G_i,G_j) est un lien dans le réseau overlay de haut niveau (X,U), alors s_i connaît le nom d'au moins un autre superpair $s_j \in S_j$. Ainsi, s_i peut envoyer des requêtes à s_j. Par contre, si p est un pair normal, p doit d'abord envoyer des messages au superpair de son groupe, lequel peut à son

tour envoyer la requête à un autre groupe. Les pairs normaux doivent alors connaître le nom et l'adresse IP des superpairs dans son groupe. La Figure 2.1(a) montre le réseau overlay de haut niveau. La Figure 2.1(b) montre des relations de communication possibles entre les superpairs correspondants. La Figure 2.2 montre un exemple où il y a un superpair par groupe et le réseau overlay de haut niveau est un anneau.

À l'intérieur de chaque groupe il y a aussi un réseau overlay qui est utilisé pour envoyer des requêtes entre les pairs dans le groupe. Chaque groupe fonctionne de façon indépendante des autres groupes. Par exemple, certains groupes peuvent utiliser Chord, d'autres CAN, ou Pastry.

2.1.1 Service de Recherche Hiérarchique

Considérons d'abord un service de recherche à deux niveaux, où le haut niveau est en charge des "pairs groupe" est celui d'en bas de "pairs noeud". Étant donnée une clé k, nous disons qu'un groupe G_j est responsable de la clé k si G_j est l'ID du groupe le plus proche de k parmi tout les groupes. Ici, "plus proche" est défini selon le service spécifique de recherche de haut niveau (e.g., Chord, CAN, Pastry, ou Tapestry). Le processus de recherche est donc :

1. En utilisant le réseau overlay du groupe i, le pair p_i envoi une requête à l'un de superpairs dans S_i.
2. Une fois que la requête atteint le superpair, le service de recherche de haut niveau route la requête à travers (X,U) vers le groupe G_j responsable de la clé k. Pendant cette phase, la requête passe exclusivement à travers des superpairs. Un superpair utilise les IP des superpairs qu'il connaît pour renvoyer la requête d'un groupe à l'autre. Finalement, la requête arrive à un superpair $s_j \in G_j$.
3. En utilisant le réseau overlay dans le groupe j, le superpair s_j route la requête vers le pair $p_j \in G_j$ responsable de la clé k.
4. Le pair p_j répond au pair à l'origine de la requête p_i. Selon le design, la réponse peut suivre le chemin inverse que vient de traverser la requête ou elle peut être envoyée directement à p_i (sans passer par les overlays).

Cet approche peut être généralisée pour un nombre arbitraire de couches. Une requête est d'abord routée à travers le réseau overlay du plus haut niveau, jusqu'à un superpair du niveau immédiatement inférieur, qui la route dans son réseau overlay "local", et ainsi jusqu'à que la requête arrive au niveau le plus bas.

L'architecture hiérarchique a plusieurs avantages quand on la compare avec les réseau overlay horizontaux.

– *Exploitation de l'hétérogénéité des pairs :* Si on choisit comme superpairs les pairs qui sont connectés le plus au système, le réseau overlay de haut niveau sera plus stable que le réseau horizontal correspondant (pour lequel il n'y a pas de hiérarchie). Cet augmentation

de le stabilité permet au service de recherche d'atteindre son efficacité optimale (par exemple, $\frac{1}{2}\log N$ en moyenne pour Chord, ou N est le nombre des pairs dans l'overlay).
– *Transparence :* Quand une clé est déplacée d'un pair à un autre à l'intérieur d'un groupe, la recherche du pair responsable de la clé est complètement transparent pour l'algorithme du réseau de haut niveau. De la même façon, si un groupe change son algorithme de recherche dans son overlay intérieur, le changement est transparent aux autres groupes. Aussi, si un pair normal $r_i \in G_i$ tombe en panne (ou si un nouveau pair apparaît) ceci sera *local* à G_i ; les tableaux de routage des pairs en dehors de G_i ne seront pas affectés.
– *Recherche plus rapide :* Parce que le nombre de groupes est typiquement des ordres de grandeur en dessous du nombre de pairs, les requêtes traversent moins des noeuds.
– *Moins de messages sur Internet :* Si les pairs les plus stables forment le THD du haut niveau, la plupart des messages pour la reconstruction des overlays vont se produire à l'intérieur des groupes, lesquels contiennent des pairs topologiquement proches. Moins de sauts pour une recherche veut dire aussi moins de messages échangés pour le même nombre de requêtes.

2.1.2 Recherche à l'Intérieur du Group

Si un groupe a un petit nombre des pairs (des dizaines), chaque pair peut tenir compte de tous les pairs dans son groupe (ses IDs et ses adresses IP) ; ce groupe pourrait utiliser CARP [72] ou *consistent hashing* [73] pour assigner et localiser des clés dans le groupe. Le nombre de pas à faire pour un tel algorithme de recherche est $O(1)$, parce que chaque pair peut déterminer avec sa liste de pairs le responsable d'une clé (G_2 dans la Figure 2.2).

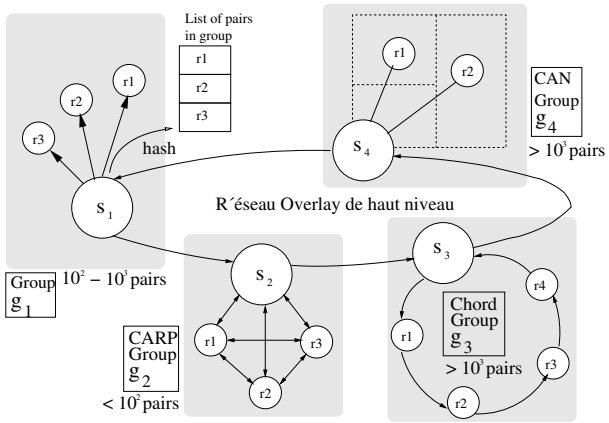


FIG. 2.2 – Un réseau overlay de haut niveau en forme d'anneau, avec un superpair par groupe. La recherche à l'intérieur des groupes se faite avec des algorithmes différents (CARP, Chord, CAN).

Si le groupe est un peu plus grand (des centaines des pairs), alors les superpairs peuvent connaître tout les pairs dans le groupe. Dans ce cas, chaque pair peut envoyer une requête au

superpair local, et celui-ci peut déterminer le pair responsable dans le groupe en $O(1)$ pas (G_1 dans la Figure 2.2).

Finalement, si le groupe est large (des milliers des pairs ou plus), alors un THD comme Chord, CAN, Pastry, ou Tapestry peut être utilisé à l'intérieur du groupe (G_3 et G_4 dans la Figure 2.2). Dans ce cas, le nombre de pas pour la recherche locale est $O(\log |G_i|)$, ou $|G_i|$ est le nombre de pairs dans le groupe.

2.1.3 Hiérarchie et Management des Groupes

Maintenant nous décrivons brièvement les protocoles utilisés pour le management des groupes : considérez le pair p qui joint le THD hiérarchique. Nous assumons que p peut obtenir l'ID G du group auquel il appartient (e.g., G peut correspondre au nom de l'ISP de p ou au campus d'une université). D'abord, p contacte un autre pair p' déjà dans le réseau P2P et lui demande de chercher la clé G. En suivant le premier pas de la recherche hiérarchique, p' trouve est renvoi l'adresse IP des superpair(s) du group correspondant. Si l'ID du groupe correspondant est précisément G, alors p joint le groupe en utilisant le mécanisme normal pour joindre le THD à l'intérieur du groupe G ; en plus, p renseigne les superpair(s) à propos de sa bande passante et sa puissance de calcul. Si l'ID du groupe n'est pas G, alors un nouveau groupe est crée avec ID G et p comme son seul (super)pair.

Dans un réseau avec m superpairs par group, les premiers m pairs à arriver au groupe G vont automatiquement devenir ses superpairs. Comme les superpairs son censés être les pairs les plus stables, nous laissons les superpairs monitoriser les pairs qu'arrivent au groupe pour identifier ceux qui restent connectés le plus long temps. Les superpairs contient une liste ordonné des candidates à superpair : le plus long temps qu'un pair a été connecté au système le plus haut qu'il est placé sur la liste. Un étude sur les réseaux non-structurés de nos jours [74] montre que le plus long temps qu'un pair a été connecté au réseau, le plus qu'il reste connecté après. Un superpair a donc besoin de mesurer combien de temps un pair reste connecté pour décider sa position sur la liste des candidats. Cet liste est envoyée périodiquement aux pairs dans le groupe. Quand un superpair tombe en panne ou se déconnecte du réseau, le premier de la liste vient le remplacer. Ce dernier s'annonce aux pairs dans le groupe ainsi que aux superpairs des groupes voisins. Si bien les groupes trop large devrait être évité, dans ces cas le *broadcast* de ces informations à tout les pairs peut être peu efficace. Au lieu de ça, les pairs peuvent apprendre petit-à-petit les changements au réseau P2P (comme par exemple, les nouveaux superpairs). Ceci peut être fait en collant les mises à jours aux réponses aux requêtes des pairs. Des algorithmes épidémiques [75, 76] peuvent être aussi utilisés pour propagé les mises à jour.

2.2 TOPLUS

Plusieurs systèmes offrant un service de recherche distribué Pair-à-Pair (P2P) ont étés récemment proposés, et tous sont basés sur des *Tableaux de Hachage Distribués (DHT)*. Pour plusieurs façons de mesurer la performance de ces systèmes – comme la vitesse de recherche ou la capacité d'implémenter du *caching* – il est très désirable que le service de recherche tienne compte de la topologie du réseau IP. Les chercheurs ont proposé de modifications aux services de recherche originaux, en prenant la topologie en considération [65, 71, 70], ou bien ont-ils créent des Overlays adaptés à la topologie [81].

Nous proposons un design de réseau P2P avec un nouveau service de recherche, le *Service de Recherche Adapté à la Topologie (Topology-Centric Look-Up Service : TOPLUS)*. Chez TOPLUS, les pairs qui sont topologiquement proches sont organisés en groupes. En plus, les groupes qui sont topologiquement proches sont organisés à son tour dans des supergroupes, et les supergroupes proches en hypergroupes, et ainsi de suite. Les groupes à chaque niveau de la hiérarchie peut être hétérogène en taille et *fan-out*. Les groupes peuvent être dérivés directement des préfixes réseau contenus dans des tableaux BGP ou des autres sources. TOPLUS a plusieurs bonnes propriétés, comme :

- *Étirement :* Nous définissons "étirement" comme la relation entre la latence entre deux points à travers le routage de couche-3 (IP) et celle en utilisant le réseau overlay. Les plus similaires que ces latences sont, le plus petit sera l'étirement. Les paquets sont routés par l'overlay vers sa destination par un chemin qui ressemble au chemin le plus court au niveau de routeurs, donnant un petit étirement.
- *Caching :* Le caching P2P des données à la demande est très facile à implémenter, et peut beaucoup réduire le temps de téléchargement des fichiers.
- *Renvoi efficace :* Comme nous verrons, les pairs peuvent utiliser des algorithmes très optimisés pour obtenir le préfixe IP le plus longue correspondant à une adresse IP pour le renvoi des requêtes.
- *Symétrique :* Le design permet aux pairs d'avoir de responsabilités similaires.

TOPLUS est un "design d'extrémiste" pour un service de recherche adapté à la topologie. Au minimum, il sert comme référence pour comparer d'autres service de recherche au niveau du étirement et de la performance du cache.

2.3 Principaux aspects de TOPLUS

Pour un message contenant la clé k, le service de recherche P2P route la requête vers le pair qui est responsable de k en ce moment. Le message voyage depuis le pair source p_s, à travers d'une série de pairs intermédiaires $p_1, p_2, \ldots, p_v$, et finalement arrive au pair destination, p_d.

Les principaux buts de TOPLUS sont : (1) Pour un message avec la clé k, le pair source p_s envoi un message (à travers de routeurs de niveau IP) jusqu'à un premier pair p_1 qui est "topologiquement proche" de p_d ; (2) Après être arrivé à p_1, le message reste topologiquement proche de p_d pendant qu'il est routé de plus en plus proche de p_d à travers des pairs intermédiaires suivants. Clairement, si le service de recherche remplies ces deux buts, l'étirement doit rester assez proche de 1. Maintenant nous décrivons TOPLUS dans le contexte d'IPv4.

Laissez I être l'ensemble de toutes les adresses IP de 32 bits[1]. Laissez $\mathcal{G}$ être une collection d'ensembles tels que $G \subseteq I$ pour chaque $G \in \mathcal{G}$. Ainsi, chaque ensemble $G \in \mathcal{G}$ est un ensemble d'adresses IP. Nous appelons chacun de ces ensembles G un *group*. Tout groupe $G \in \mathcal{G}$ qui ne contiens pas un autre groupe est nommé *groupe intérieur*. Nous disons que la collection $\mathcal{G}$ est *proprement construite* si elle satisfait les propriétés suivantes :

1. $I \in \mathcal{G}$.
2. Pour tout pair de groupes dans $\mathcal{G}$, les deux groupes sont soit disjoints, ou un groupe est un sous-ensemble de l'autre.
3. Pour chaque $G \in \mathcal{G}$, si G n'est pas un group intérieur, alors G est l'union d'un nombre fini d'ensembles appartenant à $\mathcal{G}$.
4. Chaque $G \in \mathcal{G}$ contiens un ensemble d'adresses IP consécutifs qui peut être représenté par un préfixe IP de la forme $w.x.y.z/n$ (par exemple, $123.13.78.0/23$).

La collection d'ensembles $\mathcal{G}$ peut être crée à partir de préfixes de réseau IP obtenus des tableaux BGP et des autres sources [85, 86]. Dans ce cas là, plusieurs des ensembles $\mathcal{G}$ pourraient correspondre à des AS, et d'autres à des agrégations des AS. Cet approche selon lequel on défini $\mathcal{G}$ à partir des tableaux BGP nécessite de la création d'un arbre proprement construit. Pour réduire la taille des tableaux de routage des pairs, des groupes artificiels peuvent être introduits. Remarquez que les groupes ont des tailles différentes, ainsi que des *fan-out* très divers.

Si $\mathcal{G}$ est proprement construite, alors la relation $G \subset G'$ défini un ordre partiel sur les ensembles dans $\mathcal{G}$, et elle génère un arbre d'ordre partiel avec plusieurs couches. L'ensemble I est à la couche-0, la plus haute. Un groupe G appartient à la couche-1 s'il n'existe pas un G' (autre que I) tel que $G \subset G'$. Les couches restantes sont définies de façon récursive de la même manière (voir la Figure 2.3).

2.3.1 Information par Pair

Laissez L dénoter le nombre des couches d'un arbre TOPLUS, laissez aussi U être l'ensemble de pairs disponibles et considérez un pair $p \in U$. Le pair p est contenu dans une collection d'ensembles appartenant à $\mathcal{G}$; dénotez ces ensembles par $H_i(p), H_{i-1}(p), \cdots, H_0(p) = I$,

[1] Par simplicité, nous assumons que toutes les adresses IP sont valides. Bien entendu, certains bloques d'adresses IP sont privés et d'autres ne sont pas définis. TOPLUS peut être redéfini selon ces contraintes.

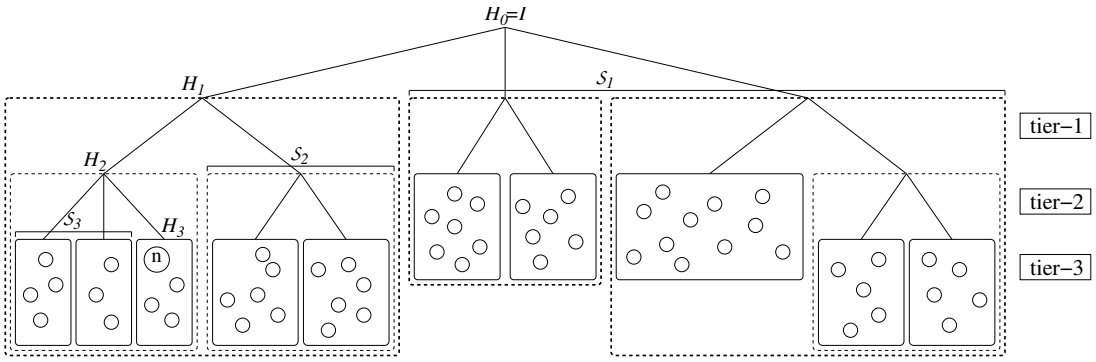


FIG. 2.3 – Un exemple de hiérarchie TOPLUS (les groupes intérieurs sont représentés par de boites pleines).

ou $H_i(p) \subset H_{i-1}(p) \subset \cdots \subset H_0(p)$ et $i \leq L$ est la profondeur en couches du groupe intérieur de p. Exception faite de $H_0(p)$, chacun de ces ensembles a un ou plusieurs jumeaux dans l'arbre d'ordre partiel (voir la Figure 2.3). Laissez $\mathcal{S}_i(p)$ être les groupes jumeaux de $H_i(p)$ à la couche i. Finalement, laissez $\mathcal{S}(p)$ être l'union des ensembles jumeaux $\mathcal{S}_1(p), \cdots, \mathcal{S}_i(p)$.

Pour chaque groupe $G \in \mathcal{S}(p)$, le pair p doit connaître l'adresses IP d'au moins un pair en G et de tout les autres pairs dans le groupes intérieur de p. Nous appelons cet collection d'adresses IP le *tableau de routage* de p, qui constitue l'information gardée par p. Le nombre total d'adresses IP dans le tableau de routage d'un pair dans la couche-i est $|H_i(p)| + |\mathcal{S}(p)|$.

2.3.2 Métrique XOR

Chaque clé k' doit être un élément d'I', ou I' est l'ensemble de toutes les chaînes binaires de b bits ($b \geq 32$ est fixé). Une clé peut être obtenue de façon aléatoire uniforme de I'. Pour une clé $k' \in I'$, dénotez k les suffixe de 32 bits de k' (donc $k \in I$ et $k = k_{31}k_{30}\ldots k_1k_0$). Pendant la discussion qui se suit, on fera plutôt référence à k que à l'originelle, k'.

La métrique XOR définit-elle une distance entre deux ID j et k comme $d(j,k) = \sum_{\nu=0}^{31} |j_\nu - k_\nu| \cdot 2^\nu$. La métrique $d(j,k)$ a les propriétés suivants :

- Si $d(i,k) = d(j,k)$ pour n'importe quelle k, alors $i = j$.
- $\max d(j,k) \leq 2^{32} - 1$.
- Laissez $c(j,k)$ être le nombre de bits dans le préfixe commun entre j et k. Si $c(j,k) = m$, $d(j,k) \leq 2^{32-m} - 1$.
- Si $d(i,k) \leq d(j,k)$, alors $c(i,k) \geq c(j,k)$.

$d(j,k)$ est une raffinement de la correspondance du préfixe le plus long. Si j est l'unique préfixe le plus long correspondant à k, alors j est le plus proche de k en termes de cet métrique. En plus, si deux pairs partage le même préfixe commun avec la clé, la métrique décidera qui est le plus proche. Le THD Kamdelia [89] utilise la métrique XOR aussi. Le pair p' qui minimise $d(k,p)$, $p \in U$ est "responsable" de la clé k.

2.3.3 L'Algorithme de Recherche

Supposons que le pair p_s veut chercher la clé k. Le pair p_s détermine le pair dans son tableau de routage qui est le plus proche de k selon la métrique XOR, disons p_j. Alors p_s envoi le message à p_j. Le processus continue jusqu'à que le message avec la clé k atteint le pair p_d tel que le pair le plus proche de k dans le tableau de routage de p_d est p_d lui même. Trivialement, p_d est le pair responsable de k.

Si l'ensemble de groupes est proprement construit, alors il est immédiat de montrer que le nombre de sauts dans une recherche est au maximum $L+1$, ou L est la profondeur, en couches, de l'arbre d'ordre partiel. Avec le premier saut le message est envoyé au pair p_1 qui est dans le même group, disons G, que p_d. Le message reste à l'intérieur de G jusqu'à qu'il arrive à p_d.

Chaque pair dans TOPLUS imite un routeur dans le sens ou le pair route le message sur la base d'une généralisation de l'algorithme du plus long préfixe correspondant à une adresse IP, en utilisant des algorithmes très optimisés [90].

2.3.4 Maintenance de l'Overlay

Quand un nouveau pair p se joint au système, p demande à un pair arbitraire de déterminer (en utilisant TOPLUS) le pair le plus proche de p (en utilisant l'adresse IP de p comme clé), dénoté par p'. p remplies son tableau de routage avec celui de p'. Le tableau de routage de p devrait alors être modifié pour satisfaire une propriété de "diversité" : pour chacun de pairs p_i dans son tableau de routage, p demande à p_i de donner un autre pair au hasard dans le même groupe que p_i. De cet manière, pour chaque deux pairs dans un groupe G, les ensembles de délégués respectives dans les autres groupes seront différentes (avec grande probabilité). Ceci assures que, si l'un de délégués tombe en panne, il est possible d'utiliser le délégué d'un autre pair. Finalement, tout les pairs dans le groupes intérieur de p doivent mettre à jour ses tableaux de routage.

Les groupes, étant virtuels, ne tombent pas en panne ; seulement les pairs peuvent le faire. Les groupes existant peuvent être partitionnés ou agrégés doucement au long du temps, selon les besoins du moment. Les clés peuvent être transférées d'un groupe à l'autre doucement : quand un pair reçoit une requête pour une clé qu'il ne stocke pas, le pair lui même peut envoyer une requête qui exclut son propre groupe. Une fois que la clé est récupérée, les requêtes ultérieurs peuvent être normalement satisfaites.

2.3.5 Design Hiérarchique

TOPLUS est un exemple de réseau P2P hiérarchique, comme on les a décrits précédemment. Un arbre TOPLUS de L couches corresponds à un THD hiérarchique de L-couches, avec un design symétrique (voir la Section 2.1), ou tous les pairs sont des superpairs. Dans chaque couche (de la première à la plus basse) un THD basé sur CARP [72] utilisant la métrique XOR est employé pour résoudre le groupe responsable d'une clé. Au dernier pas de la procédure de recherche, le même algorithme est utilisé pour trouver le pair responsable d'une clé dans le groupe intérieur de destination.

Les superpairs ne sont pas utilisés de façon *explicite*, mais il y a une recherche implicite des pairs les plus stables à travers de la procédure de conservation de la diversité : si un pair p se connecte à un délégué instable dans le groupe G, le délégué finira par abandonner le groupe assez rapidement, et donc p se verra forcé à demander un nouveau délégué dans le même groupe. Pair p changera de délégué jusqu'à qu'il trouve un que soit stable. Les pairs que depuis le debout se sont connectés à des pairs stables n'ont de raison de changer de délégués. Donc, à la fin, les pairs le plus stables jouent un rôle tacite de superpairs, mais en évitant l'algorithme de sélection qu'on a présenté dans la Section 2.1.3. Encore une fois, on voit comment les réseaux P2P ont une tendance à se construire au tour des pairs les plus stables [25].

2.4 Évaluation de TOPLUS

Les préfixes de réseau IP sont obtenus de plusieurs sources : Tableaux BGP fournis par Oregon University [93] (123,593 préfixes) et par la University of Michigan et Merit Network [94] (104,552) ; Des préfixes de réseaux IP de registres de routage fournis par Castify Networks [95] (143,082) et RIPE [96] (124,876). Après avoir mis ensemble toutes ces informations est avoir éliminés les préfixes réservés et non-routables, nous avons obtenus un ensemble de *250,562* préfixes différentes, qui nous organisons dans un arbre d'ordre partiel (nommé Arbre de Préfixes à partir de maintenant). Des représentation valide de l'Internet en 2003.

2.4.1 Évaluation de l'Étirement

L'étirement est défini comme la relation entre la latence moyenne du routage TOPLUS (en utilisant l'Arbre de Préfixes) et la latence moyenne du routage IP. Idéalement on pourrait utiliser `traceroute` [98] pour mesurer le délai entre deux hôtes arbitraires sur Internet. Mais les mesures de sécurité entreprises pratiquement sur touts les routeurs dans l'Internet nous empêchent d'utiliser cet technique simple et précise. Nous avons utilisé `king` [99] pour obtenir des résultats expérimentaux. `king` donne une bonne approximation de la distance entre deux machines quelconques, mesurant la latence entre ses serveurs DNS correspondants.

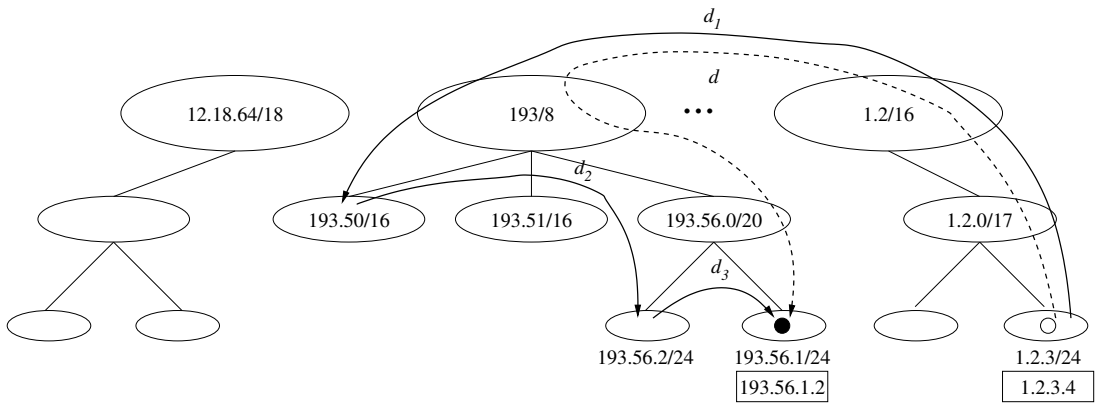


FIG. 2.4 – Chemin suivi par une requête dans l'Arbre de Préfixes.

Le principe général suivi dans nos mesures est montré dans la Figure 2.4, ou le pair avec l'adresse 1.2.3.4 envoi une requête pour la clé k, dont le pair responsable est 193.56.1.2. Suivant la procédure de routage de TOPLUS, le pair 1.2.3.4 doit d'abord envoyer la requête dans le groupe de la couche-1 qui contiens la clé k. Le pair 1.2.3.4 choisi un délégué de son tableau de routage dans le groupe $193/8$, parce que $k/32 \subset 193/8$ (remarquez qu'il ne peut pas avoir un autre groupe G dans la couche-1 satisfaisant $k/32 \subset G$). Étant donné que le délégué est dans le groupe $193.50/16$, la requête est d'abord routé à travers le chemin marqué avec latence d_1 dans la Figure 2.4. Après, le délégué choisit le group dans la couche-2 (le seul et unique) dans $193/8$ qui contient k : $193.56.0/20$. Laissez le nouveau délégué être dans le group $193.56.2/24$ de la couche-3. La requête est envoyée à ce pair à travers le chemin d_2. Finalement, le groupe de destination $193.56.1/24$ est atteint avec le saut suivant d_3. La requête aurait suivi le chemin d entre 1.2.3.4 et 193.56.1.2 avec le routage IP. L'étirement pour ce cas particulier serait donc $\frac{d_1+d_2+d_3}{d}$.

Arbre de Préfixes Originel

L'Arbre de Préfixes Originel est l'arbre résultant d'ordonner les préfixes IP en utilisant l'opérateur $\subset$. L'arbre d'ordre partiel a 47,467 groupes (préfixes) différents dans la couche-1 ; moins un sous-groupe dans la couche-3. Le nombre des groupes (non-intérieurs) à chaque couche diminue rapidement avec la profondeur de la couche, et l'arbre résultant est fortement déséquilibre.

Nous avons calculé un étirement moyen de *1.17* pour l'Arbre de Préfixes Originel, c'est-à-dire, une requête dans TOPLUS prends en utilisé le routage direct IP. Dans le Tableau 2.1 nous présentons l'étirement moyen observé pour des destinations dans la couche-1 jusqu'à la 4. Comme nous l'espérions, le plus profonde dans l'arbre que se trouve la destination, la plus grand l'étirement (parce que la probabilité de faire plus de sauts augment-elle aussi). Plus de la moitié de requêtes ont comme destination un pair dans un groupe de la couche-2.

	étirement TOPLUS vs. IP (± intervalle de confiance)		
Couche	Originel	16-bit regroup.	8-bit regroup.
1	1.00 (±0.00)	1.00 (±0.00)	1.00 (±0.00)
2	1.29 (±0.15)	1.32 (±0.14)	1.56 (±0.23)
3	1.31 (±0.16)	1.30 (±0.17)	1.53 (±0.23)
4	1.57 (±0.50)	1.41 (±0.20)	1.56 (±0.50)
Moyenne	1.17 (±0.06)	1.19 (±0.08)	1.28 (±0.09)

TAB. 2.1 – Étirement obtenu pour chaque arbre, selon la couche du pair de destination.

Arbres de Préfixes Modifiés

Comme nous avons mentionné précédemment, une grande partie de groupes se trouve dans la couche-1 et tous les pairs dans le réseau doivent connaître un délégué dans chacun des groupes. Pour réduire la taille des tableaux de routage, nous modifions l'arbre par l"'agrégation" de petits groupes qu'ont des longs préfixes et de groupes plus grands qui ne sont pas présents dans nos sources de préfixes IP. Nous considérons les groupes "petits" si ses préfixes ont une longueur de plus de 16 bits ; ceci représente 38,966 groupes dans la couche-1 de nos données expérimentaux.

Si nous regroupons les groupes petits dans de nouveaux groupes de 16-bits, nous appelons cela "regroupement 16-bit". Si les nouveaux groupes sont de 8-bit, on l'appel "regroupement 8-bit".

Les moyennes obtenues sont dans le Tableau 2.1). Ces résultats remarquables démontre que, même après une agrégation assez agressif, la propriété de petit étirement de TOPLUS est préservée.

2.4.2 Taille du Tableau de Routage

La principale motivation derrière le regroupement de préfixes est la réduction de la taille des tableaux de routage. Nous estimons la taille moyenne des tableaux de routage pour 5,000 adresses (pairs) prises au hasard de façon uniforme ; pour chacun de ces pairs p, nous examinons la structure de l'arbre pour déterminer les groupes jumeaux $\mathcal{S}(p)$ est le nombre des pairs dans le groupe intérieur $H_N(p)$, est nous pouvons calculer la taille $|\mathcal{S}(p)| + |H_N(p)|$ du tableau de routage de p.

Le Tableau 2.2 la taille moyenne du tableau de routage selon la couche à laquelle le pair appartient. La taille du tableau de routage est déterminé principalement par le nombre de groupes dans la couche-1. Si nous éliminons ces délégués du tableau de routage, la taille de ledit tableau dont on a besoin pour router des requêtes *à l'intérieur* de chaque groupe de la couche-1 reste

petite. Pour réduire encore la taille des tableaux de routage, nous transformons les arbres *Originel* et *regroupement 16-bit* de façon à que touts les groupes de la couche-1 soient des préfixes d'une longueur de 8-bits. On les appel, respectivement, *Originel+1* et *16-bits+1*.

Couche	Taille moyenne du tableau de routage						Taille moyenne du tableau de routage		
	Originel		16-bit regroup.		8-bit regroup.		Originel+1	16-bit+1	3-Couches
1	47,467	0	10,709	0	8,593	0	143	143	143
2	47,565	98	10,802	93	8,713	120	436	223	248
3	47,654	187	10,862	153	8,821	228	831	288	261
4	47,796	329	11,003	294	8,950	357	1,279	428	-
5	47,890	423	11,132	423	9,016	423	696	556	-

TAB. 2.2 – Taille moyenne du tableau de routage dans chaque arbre selon la couche ou se trouve le pair. Pour les entrées avec deux colonnes, celle de gauche est la taille du tableau de routage entier, et celle de droite est la taille sans les groupes de la couche-1.

Finalement nous créons un dernier arbre nommé *3-Couches* qui n'a que 3 couches. La couche d'en haut est formé par au plus 256 groupes avec des préfixes de 8-bits, la couche deux par au maximum 256 groupes chacun avec des préfixes d'une longueur de 16-bits, et une troisième couche avec des groupes de 24-bits. Comme l'on peut observer dans le Tableau 2.4.2, il y a un compromis entre la latence de la recherche est les besoins de mémoire pour le routage [32].

	Originel+1	16-bit+1	3-Couches
étirement (TOPLUS/IP) / marge de confiance	1.90 / ($\pm$0.20)	2.01 / ($\pm$0.22)	2.32 / ($\pm$0.09)

TAB. 2.3 – L'étirement TOPLUS vs. IP dans les arbres ou tous les groupes dans la couche-1 ont de préfixes de 8-bits.

Chapitre 3

THD Hiérarchique : Applications

3.1 Distribution de Contenues

Le multicast IP aurait pu être (ou, du moins, il était désigné pour en devenir) la solution idéal pour la distribution de contenues sur Internet : (1) Il peut servir de contenues à un nombre illimité de destinations, et (2) utilise la bande passante du réseau de façon très efficace. Ces deux caractéristiques sont en profonde corrélation. Le multicast IP épargne beaucoup de bande passante parce que un seul flux de données peut alimenter plusieurs récipients. Le flux de données n'est divisé que aux routeurs ou l'on peut trouver des destinations pour les données à travers de deux ou plus ports sortants. Ainsi, *n* clients n'ont pas besoin de *n* flux indépendants, ce que permet le passage à l'échelle du multicast IP. Pourtant, le multicast IP n'a jamais été largement déployé sur Internet : des raisons de sécurité (e.g., n'importe quelle machine peut envoyer des données vers un group multicast sans en faire partie), le manque de contrôle de congestion [101] (ce que fait la réception des donnés par des récepteurs hétérogènes difficile), ont empêché le déploiement universel du multicast IP. Il y a, pourtant, un service multicast offert par Sprint, qui reste tout même une exception.

Plusieurs propositions de multicast au niveau applicatif sont apparus [103, 104, 105, 106, 62], la plus part d'entre elles implémentées sur des infrastructures P2P (Chord [33], CAN [40] ou Pastry [41]). Le bon passage à l'échelle des réseaux P2P en dessous donne à ces multicast de niveau d'application l'une des propriétés du service originel de multicast IP, celle de pouvoir servir des contenues à un grand nombre des clients (pairs). Par contre, ces réseaux P2P sont en général conçus comme des systèmes de la couche application complètement isolés du réseau IP.

Ainsi, les systèmes multicast P2P pourraient échouer au deuxième but du multicast IP, son efficacité : un réseau local contenant un certain nombre de pairs dans un arbre multicast P2P pourrait trouver son liens de connexion à Internet saturé par de flux de données *identiques*, sauf si les pairs sont d'une façon quelconque *conscients* du fait qu'ils partagent le même réseau.

Nous avons basé notre protocole de multicast P2P sur TOPLUS parce qu'il est *conscient de la topologie* du réseau. Nous voulons construire un réseau P2P multicast qui évite la saturation des liens tout en restant capable de passer à l'échelle.

3.1.1 Un Arbre Multicast

Dans un premier approche nous assumons que tous les pairs, la source de l'arbre multicast inclue, sont connectés à travers de liens ou la bande passante n'est pas une contrainte, c'est-à-dire, il y a assez de bande passante pour l'application. Un arbre multicast simple peut être observé dans la Figure 3.1. Laissez S être la source du groupe multicast m. Bien évidement, la source doit être quelque part dans l'arbre TOPLUS, qui couvrirai la totalité de l'Internet. Le pair p reçoit le flux du pair q. On dit que q est le parent de p dans l'arbre multicast. De la même façon, on dit que p est l'enfant de q. Le pair p est au niveau-3 de l'arbre multicast et q au niveau-2. C'est important de remarquer que le niveau ou un pair se trouve dans l'arbre multicast n'a, en principe, rien à voir avec la couche ou il est dans l'arbre TOPLUS.

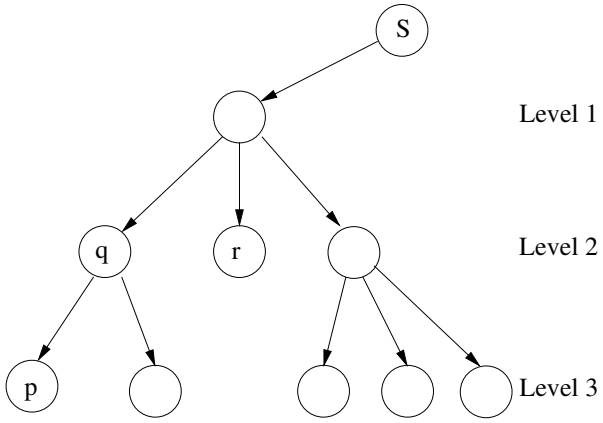


FIG. 3.1 – Un arbre multicast simple.

Pour le genre d'arbres multicast que nous voulons construire, chaque pair doit être à proximité de son parent, en terme de délai sur le réseau, au même temps qu'il essaie de joindre l'arbre multicast aussi haut (proche de la source) que possible. Chaque pair essaie quand il joint l'arbre de minimiser le nombre de saut depuis la source, et la latence du dernier saut.

3.1.2 Construction d'Arbres Multicast

Nous utilisons le réseau TOPLUS et son algorithme de recherche pour construire les arbres multicast. Considérez une adresses IP multicast m, et la clé correspondante que, en abusant de la notation, nous allons dénoté aussi m. Chaque groupe G_i de la couche-i est défini par un préfixe de réseau IP a_i/b ou a_i est un adresse IP est b est la longueur du préfixe en bits. Laissez m_i être

la clé résultante de la substitution des premiers b bits de m par ceux de a_i. Le groupe intérieur qui contiens le pair responsable de m_i (obtenu avec l'algorithme de recherche de TOPLUS) et le *group intérieur responsable*, ou GIR, pour m dans G_i (remarquez que ce GIR est contenu dans G_i). Nous dénotons, pour un pair p et une clé m, le GIR dans $H_i(p) \in$ couche-i simplement comme GIR-i de p. Ce GIR est le point de rendez-vous pour touts les pairs dans $H_i(p)$. Le plus profonde que la couche-i du GIR-i est dans l'arbre TOPLUS, le plus petit est le cercle de pairs qui l'utilisent comme rendez-vous.

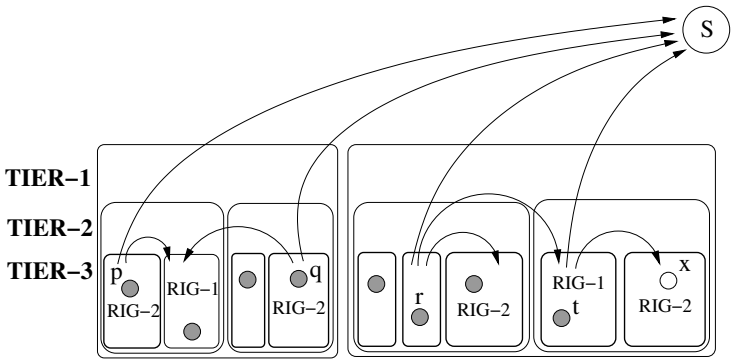
Dans l'exemple très simple de 3-couches de la Figure 3.2(a), nous avons mis les noms des différentes GIRs d'un groupe multicast donné (les pairs grises appartiennent au groupe multicast), ou touts les groupes intérieurs sont à la couche-3. Le GIR-i d'un pair peut être trouvé en suivant les flèches. Les flèches représentent le processus de demande aux GIRs d'un parent dans l'arbre multicast. Par exemple, p et q partage le même GIR-1 parce qu'ils sont dans le même groupe de la couche-1. Le groupe intérieur de t est son propre GIR-1, mais t devrait d'abord contacter le pair x (blanc) dans son GIR-2 pour lui demander un parent (si personne ne reçoit pas le flux dans le group intérieur de t, évidement). Remarquez que ce pair x n'est pas dans l'arbre multicast (Figure 3.2(b)). Certains groupes intérieurs peuvent être des GIRs pour plusieurs groupes dans de couches plus hautes. Si un pair est seul dans son groupe de la couche-1, alors il se connecte à la source.

Pensez à un pair p dans la couche-$(i+1)$ (i.e., un pair dont le groupe intérieur est dans la couche-$(i+1)$ de l'arbre TOPLUS) qui veut joindre un arbre multicast avec une adresse multicast m, que nous appelons groupe m. L'algorithme en bas peut être suivi à travers du pseudo-code dans les Figures 3.3 et 3.4.

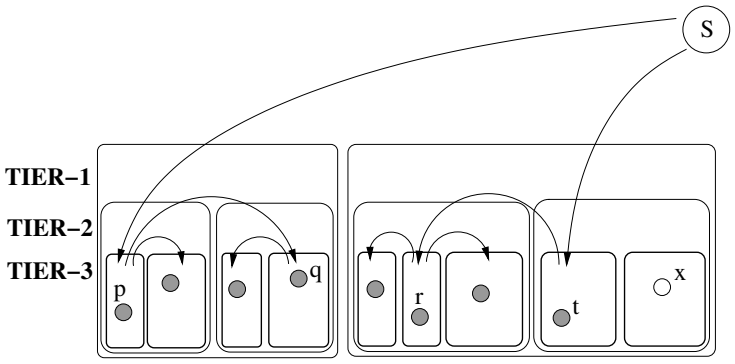
1. Le pair p broadcast une requête pour joindre le groupe m dans son groupe intérieur. S'il a déjà un pair p' qui fait partie du groupe m, p se connecte à p' pour recevoir les données (Figure 3.3, ligne 3).

2. Si un tel pair p' n'existe pas, p doit chercher son GIR-i. La recherche de m_i dans le groupe de la couche-i ou p est placé (donc parmi les groupes jumeaux de celui de p à la couche-$(i+1)$) trouve le GIR-i responsable de m. p contacte alors un pair p_i quelconque dans GIR-i (voir la Figure 3.3, ligne 9), et lui demande un pair dans le groupe multicast m (ligne 10). Si p_i connaît un pair p'' qui fait parti de m, il envoie l'adresse IP de p'' à p, et p se connecte donc à p''. Remarquez que p'' n'est pas forcement du groupe intérieur GIR-i. En tout cas p_i ajoute p à la liste qu'il a des pairs en train de recevoir m, et il partage cet information avec tous les pairs dans GIR-i. Si p'' n'existe pas, p continue de façon similaire avec GIR-$(i-1)$: p cherche m_{i-1} dans son groupe de la couche-$(i-1)$ (i.e., parmi les groupes jumeaux du groupe de p dans la couche-i). Ce processus est refait jusqu'à qu'un pair recevant m est trouvé, ou GIR-1 est atteint. Dans ce dernier cas, s'il n'y a toujours pas un pair en train de recevoir m, le pair p doit se connecter directement à la source du groupe multicast.

On peut observer que la recherche d'un pair est faite de bas en haut.



(a) Les GIRs dans un réseau TOPLUS.



(b) Un arbre multicast.

FIG. 3.2 – La hiérarchique TOPLUS ne détermine pas le structure de l'arbre multicast.

Property 1 *Quand un pair p dans la couche-*$(i+1)$ *joint l'arbre multicast, par construction, de tous les groupes* $H_{i+1}(p), H_i(p), \cdots, H_1(p)$ *ou p est contenu, p se connecte au pair* $q \in H_k$ *ou* $k = \max\{l = 1, \ldots, i+1 \; \exists r \in H_l$ *et r est un pair qui est déjà connecté à l'arbre multicast.*$\}$. C'est-à-dire , p se connecte à un pair dans la couche la plus profonde qui contiens p et un pair déjà connecté à l'arbre multicast.

Ce processus conduit un pair qui arrive p à trouver comme parent un pair q qui partage avec p le GIR au groupe de la couche la plus profonde qui les contiens tous les deux. Ceci assure que un nouveau pair se connecte au pair disponible le plus proche dans le réseau.

3.1.3 Topologie de l'Arbre Multicast

Tout pair dans n'importe quel groupe peut devenir un noeud de l'arbre multicast ; seulement les GIRs sont fixes, pour un groupe multicast et une structure de réseau TOPLUS. La topologie de l'arbre est donc aléatoire, et elle dépende seulement des pairs qui veulent faire partie de

fonction `trouve_parent(m)`

Require:
m = adresse multicast
p = pair cherchant un parent
1: $i \leftarrow p$.couche
2: $p' \leftarrow NULL$
/* essayer d'abord de trouver un pair p' connecté dans le groupe intérieur */
3: $p' \leftarrow p$.trouve_en_group_intérieur(m)
4: **while** $p' = NULL$ **do**
5: **if** $i = 1$ **then**
6: $p' \leftarrow$ source_multicast
7: **else**
8: $m_i \leftarrow p$.clé_préfixe(m,i)
9: $p_i \leftarrow p$.cherche(m_i)
/* p_i est responsable de m_i, donc il appartient à GIR-i */
10: $p' \leftarrow p_i$.demande_parent(m,p)
11: $i \leftarrow i - 1$
12: **end if**
13: **end while**
14: **return** p'

FIG. 3.3 – Obtention d'un parent dans l'arbre multicast.

l'arbre multicast. Pourtant, cet arbre est contraint de suivre la structure TOPLUS. Comme cela a été montré, un pair qui joint un arbre multicast va toujours se connecté à un pair qui est proche (selon la proximité TOPLUS).

En général, l'on peut assumer que pour tout groupe en réseau sur Internet il est mieux de garder la plus part du trafic *à l'intérieur* du groupe en évitant le trafic sortant : Entre ASes, le routage est les interfaces de *peering* sont faites sur la base de considérations principalement économiques, tant que à l'intérieur des ASes d'autres politiques ont lieu, plus orientées vers l'efficacité du réseau. Dans la plus part des LANs, le lien sortant vers l'Internet a une bande passante d'un ordre de magnitude inférieur au réseau lui-même. On montrera comment le multicast sur TOPLUS diminue le trafic entre réseaux.

Property 2 *Pour chaque groupe défini par un préfixe de réseau IP contenant au moins un pair connecté à l'arbre multicast, il existe seulement un flux de données entrant.*

Property 3 *Pour chaque groupe défini par un préfixe de réseau IP contenant au moins un pair connecté à l'arbre multicast, le nombre des flux sortants dans le pire de cas est limité par une constante.*

Property 4 *En utilisant le multicast sur TOPLUS, le nombre total des flux entrant et sortant*

fonction `demande_parent(m,p)`

Require:
m = adresse multicast
p = pair cherchant un parent
/∗ *pairs* contient un FIFO *pairs*$\{m\}$ pour chaque adresse multicast m ∗/

1: $p' \leftarrow$ pairs$\{m\}$.head
2: pairs$\{m\}$.tail $\leftarrow p$
3: **return** p'

(a) Demande d'un parent à un pair dans un GIR.

fonction `clé_préfixe(m,i)`

Require:
m = adresse multicast
i = couche de la recherche actuel de p
/∗ à partir de m, obtenir la clé qui mène à GIR-i ∗/

1: $pref_i \leftarrow p$.préfixe_de_couche(i)
2: $l \leftarrow pref_i$.length
3: **return** ((m AND $2^{32-l}-1$) OR $pref_i$)

(b) Obtention de la clé modifiée m_i qui mène à GIR-i.

FIG. 3.4 – Fonctions auxiliaires dans `trouve_parent`.

d'un groupe défini par un préfixe de réseau IP est limité par une constante.

La preuve est triviale à partir de deux propriétés précédentes.□

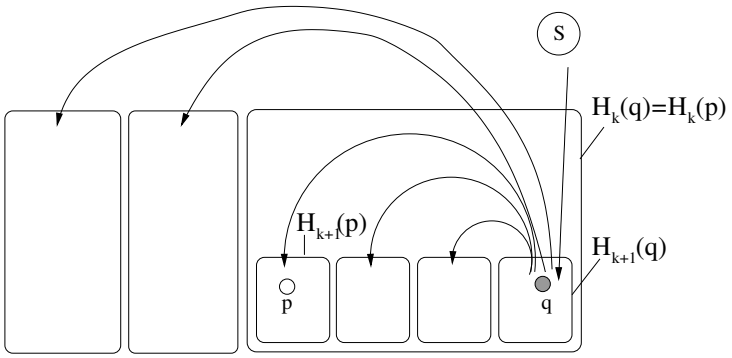
Les preuves des propriétés précédentes peuvent être trouvés sur la thèse complète. Remarquez que ces propriétés sont en principe applicables que à un environnement idéal ou un pair peut fournir autant des flux que nécessaire à ses enfants dans l'arbre multicast. Dans le monde réel, il y a une limitation à bande passante. Des différentes techniques pour gérer les situations dérivés de cet circonstance peuvent être trouver sur la thèse.

3.1.4 Algorithme de Sélection du Parent

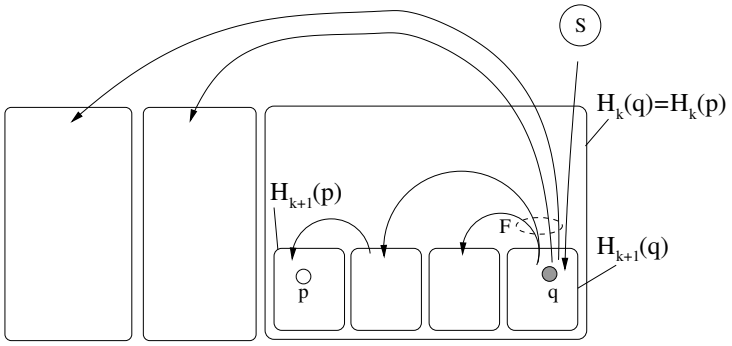
Nous avons retenus deux algorithmes de sélection du parent pour tester la construction des arbres multicast.

- FIFO, ou un pair joint l'arbre multicast au premier parent trouvé avec une connexion disponible. Quand un pair demande au GIR (en fait, ledit pair demande à un pair dans le GIR, mais nous croyons que le traitement du GIR comme une entité défini ne trompera pas le lecteur) un parent, le GIR réponde avec une liste des pairs déjà connectés. Cet liste est ordonné par temps d'arrivée au GIR. Évidement, le premier pair à arriver au GIR se connecte plus proche de la source de l'arbre multicast. Le pair arrivant teste chaque possible parent dans l'ordre des le premier jusqu'à que l'on trouve un qui accepte une connexion.
- Par Proximité, ou, *quand le premier parent de la liste a toutes ses connections occupées*, un pair se connecte au parent le plus proche qui a encore une connexion disponible.

Remarquez que MULTI+ ne vérifie pas toujours si un pair se connecte au parent le plus proche dans la liste. La raison derrière tout ça est que, si bien nous faisons confiance à TOPLUS pour trouver un parent proche, l'on préfère de se connecté à un pair haut dans l'arbre multicast



(a) Flux entrants et sortants pour chaque réseau.



(b) Flux entrants et sortants pour chaque réseau, avec $F = 4$ connections par pair.

FIG. 3.5 – Schéma qui montre les flux entrant et sortant d'un groupe.

(moins de saut vers la source) que d'optimiser le délai du dernier saut. Si MULTI+ marche correctement, la différence entre les deux algorithmes ne devrait pas être excessive, parce que le protocole prend conscience de la topologie à travers TOPLUS. Dans ces expériences on ne arrange tout le temps les pairs avec chaque arrivage, parce que le plus long qu'un pair a été connecté au système, le plus probable il est qu'il le reste [74], et les minimisations du délai à court terme pourrait affecter à long terme le stabilité du réseau.

3.1.5 MULTI+ Gérant des Pairs en Panne

Pour tester l'impact des pannes de pairs dans la performance de MULTI+, nous devons d'abord créer un arbre multicast. Nous préférons l'utilisation des données mesurés dans la réalité pour construire un environnement réaliste plutôt que utiliser un simulateur de topologies. Comme nous ne pouvons pas contrôler le comportement des algorithmes, nous devons préétablir un ensemble des pairs pour la session multicast et toutes les latences entre eux, parce que en principe, un pair peut se connecter à n'importe quel autre.

Nous concentrons nos études sur la capacité de MULTI+ à maintenir les caractéristiques des arbres multicast quand les pairs tombent en panne. Pour cet analyse nous utilisons un arbre TOPLUS *modifié* (voir la Section 2.4.1), ou nous limitons la taille des tableaux de routage par le regroupement des préfixes de réseau dans de groupes virtuels (des préfixes dans la couche-1 qui ne se trouvent pas dans les tableaux BGP).

Par la nature des systèmes P2P, ou l'infrastructure est fourni par les usages du service, les effets de l'abandonne ou des pannes des noeuds de l'arbre multicast doit être spécialement pris en considération. Nos tests analysent les propriétés des arbres multicast après l'abandonne massif contre le réseau de distribution des contenues (DoS ou virus), ou rien que des gens qui se déconnectent de la transmission d'un film décevant. Par contre, dans ce denier cas on attendrait des utilisateurs qu'ils se déconnectent proprement. Pour chaque cas, nous montrons aussi les caractéristiques des arbres multicast en absence d'erreurs, pour que le lecteur puisse apprécier les bon propriétés de MULTI+ et son niveau de résistance aux pannes.

À travers des Figures 3.6 jusqu'à 3.9 on montre la FDC (Fonction de Distribution Cumulatif) des propriétés suivantes :

- La relation R entre la latence vers le parent dans l'arbre multicast est celle vers le parent optimal (voir Figure 3.6), moins 1.
- La distribution des pairs à travers des niveaux de l'arbre multicast (voir Figure 3.7).
- Latence de la racine aux feuilles de l'arbre multicast (voir Figure 3.8).
- Nombre des flux à travers de l'interface de groupe (voir Figure 3.9).

On montre ces propriétés quand on permet un nombre illimité des connections par pair, 8 au maximum, ou seulement 4. Même s'il n'est pas réaliste, le nombre illimité de connections représente pour la plus part de propriétés de MULTI+ une limite supérieur d'efficacité. L'algorithme utilisé pour la sélection du parent est le nommé "Par Proximité", qui a donné des meilleurs résultats (voir par contre sur la thèse, que cet avantage est plutôt marginale).

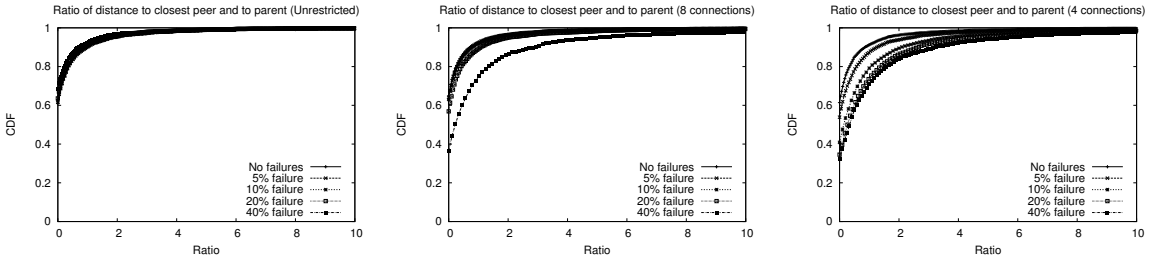
Nous faisons les remarques suivants pour en faire le résumé :

- Nous observons d'abord que quand il n'y a pas des pannes, l'algorithme travail de façon similaire quand on ne limite pas le des pairs sont connectés au parent optimal, c'est-à-dire, le parent le plus proche étant le plus haut dans l'arbre multicast. La proximité entre le parent et les enfants est plutôt préservée en présence d'erreurs (Figure 3.6). Ceci est positif, mais pas surprenant, parce que les enfants orphelins utilisent le même algorithme TOPLUS pour en trouver un nouveau parent. La dégradation dans le choix du nouveau parent est plus importante quand la restriction du nombre de connections est plus importante. Évidement, quand il n'y a pas des restrictions, les orphelins peuvent tous se connecter directement au grand parent.
- Dans la Figure 3.7 nous pouvons voir que le niveau des pairs est préservé quand le taux d'erreur est petit. Par contre, les erreurs à grande échelle poussent les pairs en bas de l'arbre jusqu'à 34 niveaux en dessous, comme pairs en panne dans la Figure 3.7(c). Le moins des connections permises par pair, le plus important l'effet pour des taux d'erreur modérés.

  Cet phénomène est expliqué facilement : Le plus grand qu'est le taux des pannes, le plus grande est la probabilité de qu'un noeud proche de la source tombe en panne. Le

plus haut qu'un pair est dans l'arbre, le plus difficile est de trouver un nouveau parent au même niveau (remarquez que le nombre de noeuds à chaque niveau de l'arbre augment de façon exponentielle avec le niveau). Dans le pire de cas, un pair directement connecté à la source, quand il tombe en panne, forces tous ces enfants (sauf celui qui prendra sa place) à se connecter aux pairs dans les feuilles de l'arbre. Ceci explique la dégradation dans les Figures 3.7(b) et 3.7(c). Ceci peut paraître inefficace, mais de cette façon là, seulement les enfants des noeuds qui tombent en panne ont à chercher de nouveaux parents, au lieu de reconstruire toute la branche.

– Du point précédente on pourrait espérer que la latence de bout-en-bout puisse se détériorer énormément quand les pairs tombent en panne. Ceci serait en effet le cas, si ce n'était par les propriétés de conscience de la topologie de TOPLUS. Dans la Figure 3.8 l'on peut voir que même si la proportion des pairs qui tombent en panne augmente, la latence ne le fait pas dans la même proportion. L'augmentation de la latence est plus importante le moins des connections que on permet par pair.
– Le nombre de flux par groupe est montré dans la Figure 3.9. Nous observons que l'utilisation de chaque réseau est base, seulement quelque flux par groupe. Ceci est par la conscience de la topologie des nos algorithmes. La structure de ces arbres multicast n'est pas beaucoup affectée par les panes des pairs, par les restrictions imposées par l'arbre TOPLUS.



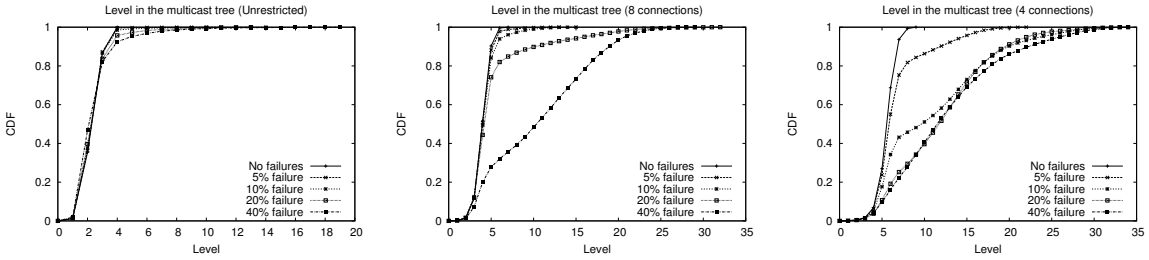
(a) Connections illimitées. (b) 8 connections. (c) 4 connections.

FIG. 3.6 – Relation de proximité entre le parent optimal et le parent choisi.

3.2 Localisation de Contenues

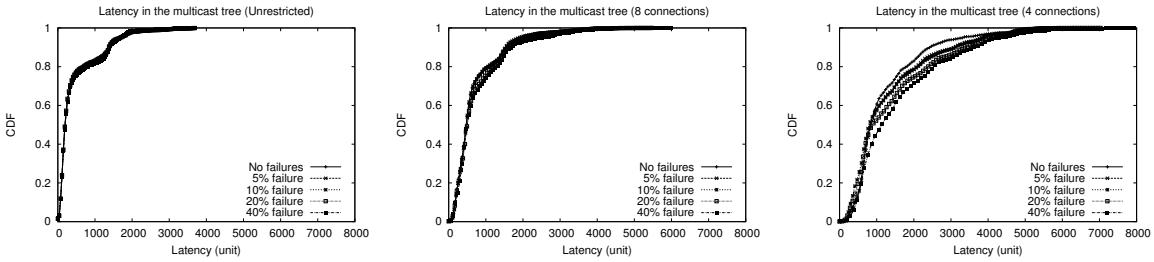
Une limitation majeur des services de recherche basés sur THD est que ils ne font que de recherches exactes : il faut connaître la clé exacte (identificateur) d'un donnée pour trouver le pair responsable de la même. En pratique, par contre, les utilisateurs de systèmes P2P n'ont d'habitude que une information partielle pour identifier les ressources, et ils ont une tendance à faire des requêtes assez vastes [27] (e.g., tous les articles écrits par "François DeCon").

Dans cet section nous proposons d'augmenter les systèmes P2P basés sur THD avec des mécanismes pour la localisation de données en utilisant des informations incomplètes. Nos mécanismes sont basés sur des indexes, stockés et distribués à travers des pairs dans le réseau,



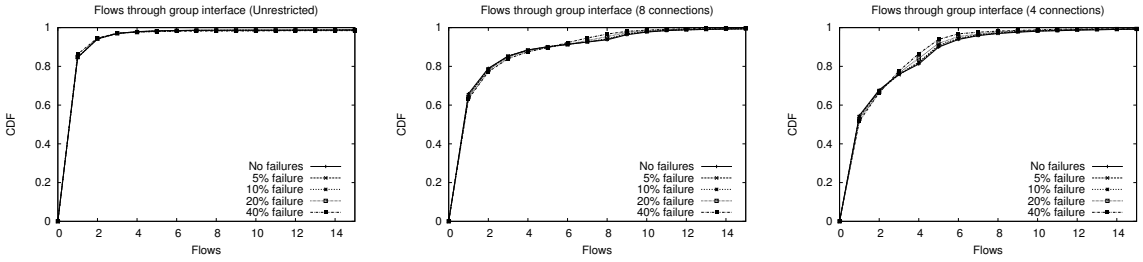
(a) Connections illimitées. (b) 8 connections. (c) 4 connections.

FIG. 3.7 – Niveau des pairs dans l'arbre multicast.



(a) Connections illimitées. (b) 8 connections. (c) 4 connections.

FIG. 3.8 – Latence de la racine aux feuilles (en unités de coordonnées TC) dans l'arbre multicast.



(a) Connections illimitées. (b) 8 connections. (c) 4 connections.

FIG. 3.9 – Nombre de flux par interface de groupe.

qui maintient des informations utiles pour les requêtes. À partir d'une requête vaste, un usager peut obtenir des informations additionnelles à propos des donnés qui corresponde à sa première requête. Le service de recherche THD peut être utilisé de façon récursive jusqu'à que l'on trouve les données désirées. Les indexes sont organisés de manière hiérarchique pour réduire l'espace de stockage est la bande passante utilisée. Le système intègre aussi un système de cache distribué pour accélérer l'accès aux contenues très demandés.

Notre technique d'indexation peut être déployée sur n'importe quelle type de THD, et bénéficier de ses propriétés (e.g., réplication, équilibrage de la charge). Nous avons entreprise une évaluation qui démontre l'efficacité de notre approche. Évidement, le temps requis pour la recherche des données dépend de la "précision" de la requête initiale : les plus vastes prennent plus de temps à trouver les données que les plus précises.

3.2.1 Algorithme d'Indexation

Le principe sous-jasent notre technique est de générer des multiple clés pour un descripteur donné, et stocker après ces clés dans des indexes organisés par le THD dans le système P2P. Les indexes ne contiennent pas des correspondances clé-à-donnée ; au lieu de ça, ils offrent un service de clé-à-clé, ou plus précisément, de requête-à-requête. Pour une requête donnée q, le service d'index répond avec une liste (peut-être vide) des requêtes plus spécifiques, couvertes par q. Si q est la requête la plus spécifique d'un fichier, alors le système de stockage P2P répond avec le fichier lui-même (ou indique le pair responsable). Au moyen des requêtes itératifs, un utilisateur peut traverser jusqu'à la racine l'arbre d'ordre partiel des requêtes et découvrir tous les fichier indexes qui correspondent à sa requête plus vaste.

Pour bien gérer les indexes, le système de stockage P2P sous-jasent doit être légèrement améliore. Chaque pair maintient un index, lequel consiste pour l'essentiel des correspondance de requête à requête. La fonction "*insrer*$(\mathsf{q}, \mathsf{q}_i)$", avec $\mathsf{q} \sqsupseteq \mathsf{q}_i$, ajoute une correspondance $(\mathsf{q}; \mathsf{q}_i)$ à l'index du pair responsable de la clé q. La fonction "*recherche*(q)", si q n'est pas la requête la plus spécifique pour un fichier, répond avec la liste de toutes les requêtes q_i telles que il y existe une correspondance $(\mathsf{q}; \mathsf{q}_i)$ dans l'index du pair responsable de la clé q.

```
<article>
  <author>
    <first>John</first>
    <last>Smith</last>
  </author>
  <title>TCP</title>
  <conf>SIGCOMM</conf>
  <year>1989</year>
  <size>315635</size>
</article>
```

d_1

```
<article>
  <author>
    <first>John</first>
    <last>Smith</last>
  </author>
  <title>IPv6</title>
  <conf>INFOCOM</conf>
  <year>1996</year>
  <size>312352</size>
</article>
```

d_2

```
<article>
  <author>
    <first>Alan</first>
    <last>Doe</last>
  </author>
  <title>Wavelets</title>
  <conf>INFOCOM</conf>
  <year>1996</year>
  <size>259827</size>
</article>
```

d_3

FIG. 3.10 – Exemples de descripteurs de fichiers.

Nous stockons les fichiers et nous construisons les indexes comme se suit : Pour un fichier f et son descripteur d, avec sa requête la plus spécifique q, le fichier est stocké dans le pair responsable de la clé $k = h(\mathsf{q})$. Un ensemble des clés $\mathsf{q} = \{\mathsf{q}_1, \mathsf{q}_2, \ldots, \mathsf{q}_l\}$ que les usagers puissent utiliser souvent est génère, telles que chaque $\mathsf{q}_i \sqsupseteq \mathsf{q}$. Après la clé numérique $k_i = h(\mathsf{q}_i)$ est obtenue pour chacune des requêtes, et on stocke les correspondances $(\mathsf{q}_i; \mathsf{q})$ dans l'index du pair responsable de chaque k_i dans le réseau P2P. Par itération de ce processus sur toutes les q_i, nous continuons de façon récursive jusqu'à que tous les indexes désirés soient obtenus.

3.2.2 Exemple d'Indexation

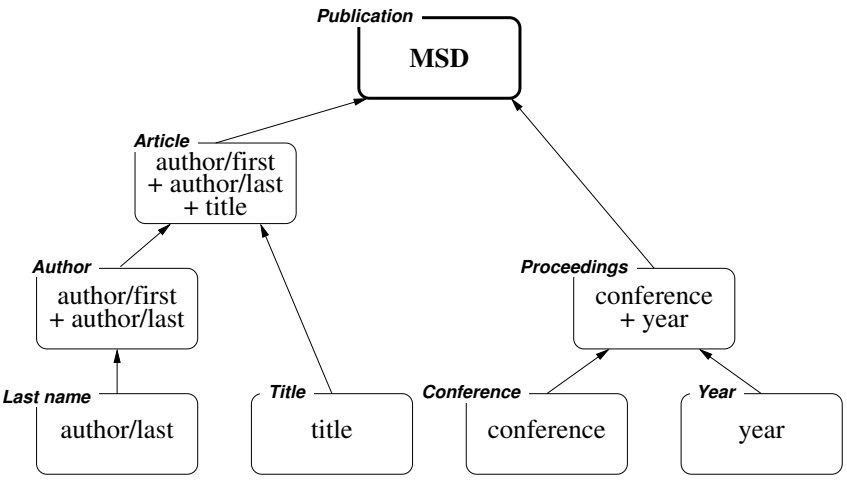


FIG. 3.11 – Exemple de schéma d'indexation pour un base de données bibliographiques.

Pour mieux illustré le principe de nos techniques d'indexation, considérez une base de données bibliographiques P2P qui stocke les trois fichiers associés aux descripteurs dans la Figure 3.10. Nous voulons pouvoir chercher les publications selon les différentes combinaisons de nom, titre, conférence et année de publication. Un possible schéma d'indexation hiérarchique est montré dans la Figure 3.11. Chaque boite corresponde à un index distribué, et les clés d'indexation sont indiquées dans les boites. L'index à l'origine d'une flèche stocke les correspondances entre sa clé d'indexation et la clé d'indexation de la cible.

Après application de cet schéma d'indexation aux trois fichier de la base de données bibliographiques, nous obtenons les indexes distribuées montres dans la Figure 3.12. L'index du niveau supérieur *Publication* corresponde aux entrées directement stockées dans le système de stockage P2P : la clé complète donne l'accès au fichier associé. Les autres indexes contiennent les correspondances de requête-à-requête qui permettent aux usagers de cherche la base de données et trouver les fichiers.

La Figure 3.13 donne les détailles des correspondances entre requêtes individuelles de la Figure 3.12. Chaque flèche montre une correspondance de requête-à-requête, e.g., $(\mathsf{q}_6; \mathsf{q}_3)$.

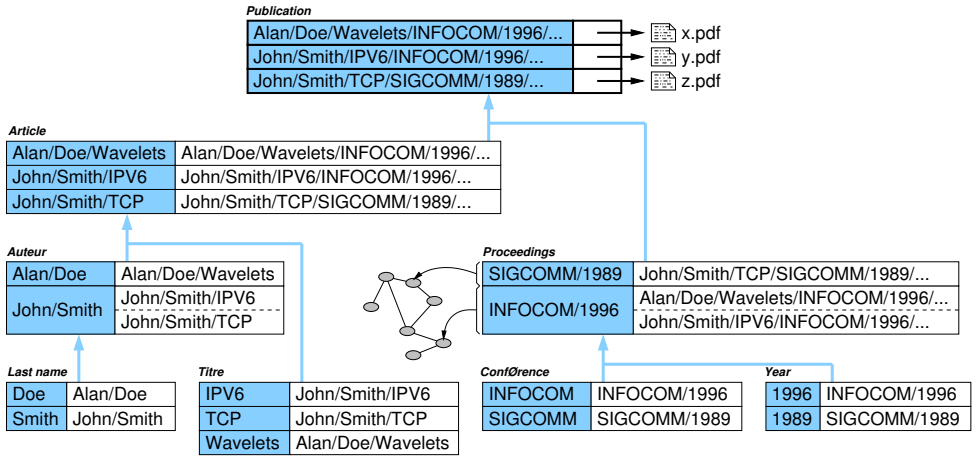


FIG. 3.12 – Exemple des indexes distribués pour les trois documents de la Figure 3.10 et le schéma d'indexation de la Figure 3.11 (la syntaxe de la requête a été simplifié).

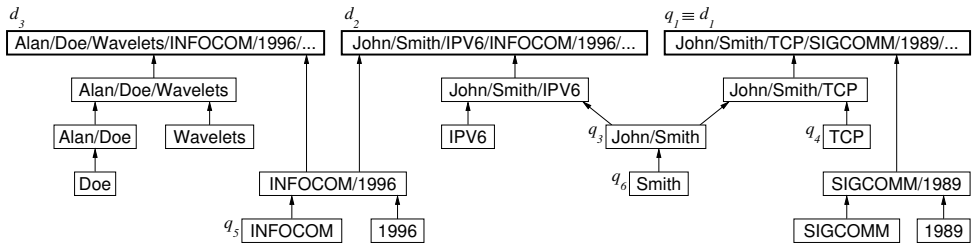


FIG. 3.13 – Correspondance des requêtes et des index pour la Figure 3.12 (identificateurs correspondent à la Figures 3.10 ; la syntaxe des requêtes a été simplifié).

3.2.3 Avantages de l'Indexation

Nous mettons en valeur ici les propriétés les plus intéressantes de la technique d'indexation :

- *Efficacité en Espace :* D'abord, comme les indexes ne contiennent que des correspondance entre clés, les données n'ont pas à être stockés sur plusieurs pairs. Deuxièmement, même si les données peuvent être atteints à travers plusieurs chemins différents, le besoin d'espace reste raisonnable parce que le indexes peu précis sont partages par beaucoup de données. (e.g., pour les correspondance de la Figure 3.13, $(q_6; q_3)$ est dans le chemin de d_1 et d_2). Finalement, l'organisation hiérarchique des indexes réduit la taille des résultats de recherches, et donc des besoins de bande passante.
- *Passage à l'échelle :* Comme les données peuvent être trouvés à travers des chemins différents et il y a des indexes différents qu'y font référence, la charge devrait être partagée entre plusieurs noeuds (en contraste avec un index centralisé). En plus, comme les indexes sont stockés comme les autres données, ils peuvent bénéficier de tous les services du système P2P pour la disponibilité et le passage à l'échelle, comme la réplication, ou le cache.
- *Faible couplage entre les données et les indexes :* Quand les données changent, seulement les pairs responsables de la clé complète doivent être mis à jour. Les indexes n'en ont pas besoin, comme conséquence de la technique de clé-à-clé.
- *Versatilité :* Certaines données peuvent ne pas être indexés, et d'autres l'être de façon très agressif pour y accélérer l'accès.
- *Adaptation :* Le système peut créer des index à la demande pour s'adapter aux habitudes des usagers ou pour offrir un service de cache.
- *Résistance au linkage arbitraire :* Quand on insère un fichier dans le système, il ne peut être indexé que aux endroits ou correspondent les clés couvrant la clé du fichier. Des linkages arbitraires vers un fichier ne peuvent pas être insérés dans le système, si la correction de la relation entre les indexes est vérifié par les pairs. Ceci fait plus difficile pour un attaquant d'introduire un fichier avec des contenues offensifs ou dangereux en le faisant passer par un autre.

Chapitre 4

Conclusions et Perspectives

Les systèmes Pair-à-Pair ont été les derniers invités à arriver à l'Internet. Ils représentent un nouveau moyen d'offrir de services aux bouts de l'Internet pour les usagers et par les usagers. Le potentiel des réseaux P2P pour créer de services nouveaux et innovateurs est à l'origine leur l'immense croissance, jusqu'à attendre des millions d'usagers. Celle-là est aussi la raison pour laquelle les systèmes P2P doivent être considérés pour développer des nouveaux services sur Internet. Mais la nature distribuée du Pair-à-Pair et donc manque de management centralisé des ressources fait que les algorithmes utilisés doivent passer à l'échelle très efficacement, ainsi que faire une bonne utilisation des ressources éparpillés parmi les très grand nombre des machines connectées à Internet.

Les systèmes P2P structurés utilisant des Tableaux de Hachage Distribués présentent des propriétés très intéressantes (passage à l'échelle pour une très grande population d'usagers, recherche des ressources très rapide) pour la Distribution et la Localisation des Contenues. Nous avons explorés ces questions par la présentation, d'abord, d'un étude d'introduction aux THDs hiérarchiques, qui a servi comme base pour le développement de notre THD, nommé TOPLUS. Les caractéristiques de TOPLUS nous ont permis de proposer un service multicast de niveau applicatif appelé MULTI+. Ses caractéristiques sont très bien adaptées pour des environnements distribués à très grande échelle, qui nécessite d'une utilisation efficace des ressources. Finalement, nous avons traité la localisation de ressources par moyen du stockage des indexes dans un réseau P2P qui peuvent être recherchés par un service de recherche THD.

Les systèmes Pair-à-Pair est un domaine en expansion, ou des nouveaux algorithmes et des nouvelles applications apparent tout les jours, ainsi que des nouveaux problèmes et challenges. À partir de notre expérience pendant les travaux de thèse que l'on vient de présenter, nous avons trouvés certains sujets pour l'avenir de notre recherche :

– Nous avons déjà vu que les THDs font correspondre une clé à un pair dans un réseau P2P, mais nous avons aussi remarqué que, selon l'application P2P, les requêtes pour les différents clés ont des fréquences très différentes. Ceci introduit un important déséquilibrasse

entre la charge des différents pairs, indépendamment de la distribution des clés entre les pairs. Les systèmes P2P basés sur THDs ont besoin des nouveaux algorithmes pour réagir de façon intelligente au comportement des usagers. Les préférences des utilisateurs ne peuvent pas être déduites *à priori*, et elles peuvent changer au cours du temps. Nous proposons que les pairs qui subissent une charge très lourde puissent créer des nouvelles répliques des clés très demandées chez des pairs choisis au hasard. Les pairs avec une charge très importante auraient besoin de modifier les tableaux de routages de ses voisins de façon temporelle pour renvoyées les requêtes vers les nouvelles localisations. Si la charge chez ces dernières restait importante, ils pourraient procéder de la même manière. Ceci produirai une redistribution des requêtes pour des clés très demandées entre les pairs qui stockent les répliques ainsi générées. Le modifications réalisées sur les tableaux de routage pourraient être défaites une fois que la charge revienne au dessous d'un certain niveau. Cet technique relève certains problèmes de sécurité, parce que certains pairs pourraient modifier des tableaux de routage pour empêcher l'accès à certaines clés.

- La sécurité chez les systèmes Pair-à-Pair est d'un énorme intérêt. Il y a un nombre important d'applications qui ne pourront jamais profiter des possibilités des réseaux P2P sauf si des mécanismes de sécurité distribués sont développés. Le design des algorithmes complètement distribués pour des services basiques de sécurité est un véritable challenge.
- Le fait que les clés chez les THDs hiérarchiques correspondent à des groupes (et à un pair dans le group, mais une clé peut être copiée dans multiple pairs à l'intérieur du group) présentent l'avantage de profiter de la plus grande stabilité des groupes, par rapport à celle des pairs. Par contre, les problèmes posent par la panne ou modification d'un groupe sont beaucoup plus importants (même s'ils n'arrivent que très rarement). Le transport des clés d'un groupe à un autre peut créer une avalanche de trafic qui peut congestionner le réseau. Nous avons présenté certains algorithmes fainéants (*lazy*) qui mettent à jour en douceur le réseau P2P quand des nouveaux groupes sont crées, mais ces approches ne seront peut-être pas assez réactifs s'il y a des pannes de réseau. Nous avons besoin d'algorithmes qui permettent le développement "organique" des réseaux P2P structurés.
- Nous avons présenté des techniques d'équilibrage de la charge entre les groupes. Pourtant, nous avons besoin d'un analyse plus approfondi et une évaluation de ces techniques et aussi d'autres, comme celles basées sur des fonctions de hachages linéales [148]. En effet, le problème du manque de corrélation entre la population des pairs dans un groupe et le nombre de clés correspondantes à ce groupe là nécessite de plus d'étude dans le cadre de THD hiérarchiques.
- Même si nos techniques d'indexation nous permettent de chercher des données avec de l'information incomplète, elles dépendent toujours d'une correspondance exacte qui se produit dans le THD dessous. Des techniques de correspondance "fuzzy" offrent de possibilités de recherche très intéressant pour traiter les requêtes ou les indexes contenants d'erreurs t'orthographie, par exemple. Ces erreurs peuvent être traités aussi par validation avec une base de données ou les indexes corrects connus sont stockés, comme CDDB [149] pour le fichiers musicaux.
- La question finale pourrait bien être : peut-on trouver une solution conciliante la rapidité de recherche de THDs et la flexibilité de réseaux P2P non-structurés ?

PhD thesis

Ecole nationale supérieure des télécommunications

Communications and Electronics department

Digital communications group

Luis Garcés-Erice

A Hierarchical P2P Network : Design and Applications

Defense date : December, the 6th 2004

Committee in charge :

Prof. Isabelle Demeure	**Chairman**
Dr. Anne-Marie Kermarrec	**Reporters**
Prof. Marcel Waldvogel	
Prof. Ernst W. Biersack	**Advisor**

Ecole nationale supérieure des télécommunications

Acknowledgments

There are many things we do not dare try because they are difficult, but they are difficult because we do not dare try them.

Seneca

Looking back at these three last years, there are many things I can thank for to lots of people. But let's get first things first. Because, I must confess, I had never had in mind getting into a PhD. Never, until one person simply told me "I think you're perfectly capable of doing it". That person is my advisor, Ernst Biersack. Now I can say that starting my PhD has been one of the best decisions in my life, and that's something I owe him. I could go on saying that it has been great to work together, that I have learned many things from him and with him, but others have said that in the past, and surely others will continue to say it in the future. I want to thank him for his encouraging words, and for the opportunity he gave me.

I cannot forget thanking Keith Ross and Pascal Felber for their innovative ideas, all that I've learned from them, and the great time that it has been to work with them. Together with Ernst we had a very motivating research team, with plenty of suggestions, questions and discussions. I believe that this has been an experience that is now a reference to me, with which all others I (hopefully) will discover will have to measure themselves.

I also have to thank Guillaume Urvoy-Keller. Granted, he has not always been a convinced supporter of P2P systems. But every time that mathematics have been in our way he has been of the greatest help, and he has always found time to patiently correct all those papers, presentations, and other documents that I had to write *en la langue de Voltaire.*

Special thanks to the members of the jury, who have greatly helped with their suggestions and remarks to improve the content of this thesis. Their time, effort and kind support are most appreciated, as much as their friendship, which I look forward to developing further in the future.

Of course, three years lets one know lots of people, and this time would have not been the same without them. I can not name them all, but I thank all those *ERASMUS* students, PhD candidates at Eurécom, I3S people, and the long list of friends from so many different places that have made life what it is, sometimes great, sometimes sad, but all the time a most interesting experience and a unique opportunity.

And last, but not least, I of course thank my family, always backing me in everything I have done and, I know, anything I will do in the future, and I thank those friends that no matter what, and no matter where, will always make me feel like being at that place called home.

¡ Gracias a todos !

Abstract

The advent of wide-spread broadband Internet connections and ever-cheaper computing devices has make it possible for Internet users to implement, offer and use networked services without the need to rely on a centralized, third party infrastructure. These systems that have emerged on the *edges* of the Internet are called Peer-to-peer (P2P) systems.

Two main large groups of P2P systems can be considered: structured and unstructured. The main difference between the two relies on the look-up method used to locate resources on the P2P network.

The structured P2P systems feature a look-up where a resource is uniquely mapped to a given peer in a given state of the system. The mapping is done by minimization of a predefined metric applied to a numerical identifier space shared by both peers and resources. The look-up process is completely deterministic, and links between peers are done following well-defined rules. The look-up infrastructure works like a Distributed Hash Table (DHT) implemented across the peers.

The unstructured P2P networks do not tightly couple peers and resources, because of the absence of a common identifier space, if any at all. The look-up procedure is random in the sense that given a resource, there is not a predefined goal as on where the search must finish. Connections between peers are random meaning that they are established following convenience or just no rule at all, but there are not predefined or necessary relationships between given peers.

In this work we focus on structured P2P systems, seduced by their novelty and the scalability properties of DHTs. Particularly, we study some aspects of the Content Distribution and Location issues, capital matters in the traditional Internet services (specifically, its main application, the World Wide Web) applied to structured P2P networks.

In our study, we have followed a bottom-up approach. We start considering DHT design issues, presenting a hierarchical P2P system framework we elaborate upon. We enumerate the hierarchical organization main characteristics and advantages: specifically, the capacity of these systems to improve networked communication performance by adapting to the Internet topology, and to increase the robustness of flat P2P systems by differentiating peer roles according to their capabilities.

Taking a particular case of the hierarchical P2P system framework studied, we develop a DHT named TOPLUS, which is built by grouping peers in the same IP Network Prefix, and

arranging these groups following the Internet hierarchical topology. We present the characteristics of TOPLUS, and we study in more detail its main goal and feature: A look-up that is able to find the peer responsible for a resource in time comparable to the level-3 routing of the resource request from the peer originating the query to the destination peer where the resource is located. Because the structure of TOPLUS follows that of the Internet, we say that TOPLUS is topology-aware.

Then we proceed to propose solutions for Content Distribution and Location in structured P2P systems.

For Content Distribution we present MULTI+, an application-level multicast P2P network that works on top of TOPLUS. MULTI+ builds multicast trees that attempt shortest possible average delay from the multicast information source to the recipients in the multicast group, while trying to use bandwidth efficiently. These goals are achieved by algorithms for multicast tree construction that take advantage of the topology-awareness of the TOPLUS DHT. We also test how robust these delay and bandwidth properties are when massive peer failure occurs.

Finally, we present a Content Location technique for structured P2P systems, where we use indexing of resources using predefined indexing schemes. Each index is used as the key to find other indexes with more precise information about a given resource, and the index completely describing the item is the key corresponding to the resource. The peer in the P2P network responsible for an index stores all indexes pointed to by the former index. This way the Content Location infrastructure is contained in the P2P system and inherits all its properties.

List of abbreviations

To improve the readability of the text, the meaning of an abbreviation or an acronym is often given only once at its first appearance in the text of a chapter. We include this list of the most often used abbreviations for the reader's reference.

AS	Autonomous System (plural: ASes)
BGP	Border Gateway Protocol
CCDF	Complementary Cumulative Distribution Function
CDF	Cumulative Distribution Function
DHT	Distributed Hash Table
DNS	Domain Name System
FIFO	First In, First Out
ID	Identifier (plural: IDs)
IP	Internet Protocol
ISP	Internet Service Provider
LAN	Local Area Network (plural: LANs)
LRU	Least Recently Used
MSD	Most Specific Descriptor (plural: MSDs)
P2P	Peer-to-Peer
RIG	Responsible Inner Group (plural: RIGs)
RTT	Round Trip Time
TCP	Transport Control Protocol
WAN	Wide Area Network (plural: WANs)
XML	eXtended Mark-up Language
XOR	eXclusive OR (bit-wise operation)
XPath	an XML-based query construction language

Notation

We have grouped here the notation used through the different chapters in this document. While possible, we have tried to maintain a uniform notation all over the text. First we present a general list and then other lists concerning the different chapters. Please note that only the notation differing from the general one appears in the chapters' lists. Finally, some seldomly used notation has been omitted.

General notation

k	A key.
$p, q, r, \ldots$	Each letter is a peer. Abusing the notation, each letter is also the ID of a peer.
$p, p', p'', \ldots$	Peers involved sequentialy in a P2P operation.
$G, G', \ldots$	Each one is a group of peers. Abusing the notation, each letter is also the ID of a group of peers.
$\|G\|$	Number of peers in group G.
N	Number of peers in a P2P system.
U	Set of peers currently available in a P2P system.
$d(p,q)$	For two IDs p and q, the distance between them according to some metric (e.g., XOR metric).
$d(p,q)$	For two peers p and q, the distance between them according to some metric (e.g., latency).
$d(G,G')$	For two groups G and G', equals to the distance $d(p,q)$ where $p \in G$ and $q \in G'$.

Chapter 2

G_i	ID of group i. Abusing the notation, the set of peers in group i.
p_i	A peer belonging to group i. Abusing the notation, the ID of the peer.
S_i	Set of superpeers in group G_i.
s_i	A superpeer in group G_i. Abusing the notation, the ID of the peer.
R_i	Set of *regular* peers in group G_i. $R_i = G_i \setminus S_i$
r_i	A regular peer in group G_i. Abusing the notation, the ID of the peer.

Chapter 3

I	Group containing all groups in a TOPLUS tree.
L	Maximum number of tiers in a TOPLUS tree.
$H_i(p), \cdots$	Set of telescoping sets containing a peer p in a TOPLUS tree.
$\cdots, H_0(p) = I$	
$\mathcal{S}_i(p)$	Set of sibling groups of $H_i(p)$.
$\mathcal{S}(p)$	$\bigcup \mathcal{S}_i(p)$.
$d^T(G, G')$	Distance between groups G and G' through TOPLUS routing path.
$d^{IP}(G, G')$	Equals $d(G, G')$, using IP latency as metric.

Chapters 4 and 5

m	A Multicast address; a Multicast group. The key identifying a Multicast group.
m_i	The key resulting of substituting its first bits by those of the IP network prefix of a group at tier-i of a TOPLUS tree.
RIG-i	*R*esponsible *I*nner *G*roup at tier-i of a TOPLUS tree. It is the group responsible for key m_i.

Chapter 6

$d, d_1, d_2, \ldots$	Each one is the descriptor of a resource.
$\mathsf{q}, \mathsf{q}_1, \mathsf{q}_2, \ldots$	Queries for resources (XPath expressions).

Chapter 1

Introduction

These are my principles. If you don't like them, I have many others.

Groucho Marx

1.1 Once Upon a Time ...

The Internet we know today might had become many things since the early days of the Arpanet. The network that one day will allow communication between any two points in the world might have been today an intelligent infrastructure, ready to fulfill its millions of users' needs [1]. Offering a myriad of services, the network would adapt to people's behavior and preferences to make its utilization seamless. Opening the doors of this computing paradise to the masses, access providers would offer exclusive content to their users through a pay-per-view service or a monthly fee to make connecting to the Internet worth [2]. All this could have happened, but it did not. Indeed, the Internet is not a million channel cable T.V. network; the Internet has no real central point of control, and those who send or offer information use the same technology as those who receive it.

The Internet today is basically the physical infrastructure and the collection of protocols (such as IEEE 802.3 Ethernet or the wireless 802.11 protocol family in the edges, and ATM or MPLS in the core) needed by Internet Protocol (IP) [3] to send information from one host to another. The Internet Protocol was designed in the beginning as a means to connect different existing networks (which used protocols like IBM's SNA or Digital's DECNET [4]), to create an internetwork. The goal of this internetwork was to allow communication between its participants, and thus data exchange among their networks became naturally *its most important function*. The protocol assuring this important function, IP, became the ubiquitous network protocol simply [5] transferring packets from one point to another in the Internet [6]. The mechanisms that process the data contained in the packets are left to the initial sender and final receiver of

those packets, following the end-to-end principle [7].

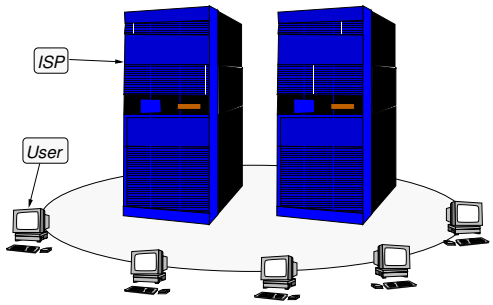
Communication involves at least two parties. The Internet, like any other means of communication, is of no interest itself. It is worth owning a house, paying for a car: an individual takes a direct benefit from these goods without the participation of others. This is not the case of a communication means. Users perceive value there because other users are participating[1] to send and receive data [8]. However, the necessary means to exchange data, namely computers and communication lines, used to be prohibitively expensive. Individual users depended on the facilities at their own Internet Service Provider (ISP) in order to receive, send (e-Mail [9], News Forums [10]), or publish (Web pages [11], FTP [12] servers) information (Figure 1.1(a)).

The more widespread usage of the Internet [13, 14, 8] created opportunities for third parties, which could offer services to users without providing the access themselves (Figure 1.1(b)). E-Mail services offered through a Web interface with remote storage, known as web-mail, and Instant person to person message services [15], as well as electronic commerce (see [16] and also [4], Section III.A) are good examples of these initiatives. Still, regular users were dependent on the infrastructure provided by those third parties, mainly because of the resources required to offer these services efficiently to a large number of users.

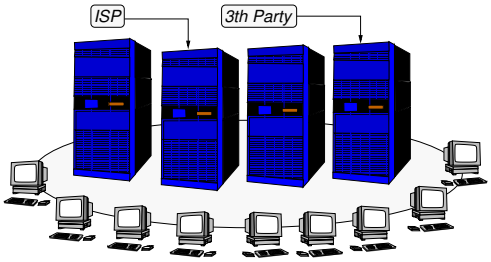
Eventually, with the PC, powerful computers became available to the masses [17]. Competition among ISPs for new customers drove prices for Internet access down, while increasing the bandwidth offered to the end users. Anyone who has a PC and communication to the Internet has now sufficient resources to provide services traditionally reserved to ISPs, like e-Mail or web page hosting (Figure 1.1(c)), a possibility facilitated by public domain software for operating systems (e.g., GNU/Linux-based software distributions) and applications (e.g., Apache Web server). Although the step from people paying money for an e-Mail address to the same people offering their friends e-Mail addresses at their own domain is impressive, in reality few regular users actually offered the traditional Internet services (Web page hosting, FTP server, etc). Even if they did so, the load that could be sustained was modest, which restricted the service to a small community.

While powerful resources are now available at the edges of the Internet, it seemed that the old services could not benefit from that development, and for a reason. These services were not designed for such a scenario. Users were supposed to *use* services, but not to *provide* them and the PCs of millions of Internet users remained idle for most of the time. They were lacking the technology that would make possible to harness the resources scattered across the Internet's endpoints. And there appeared what we call today peer-to-peer or P2P systems (Gnutella [18], Fast-Track) and applications (Napster [19], KaZaA [20], E-Mule [21], Grokster, LimeWire, Morpheus, BitTorrent [22], etc).

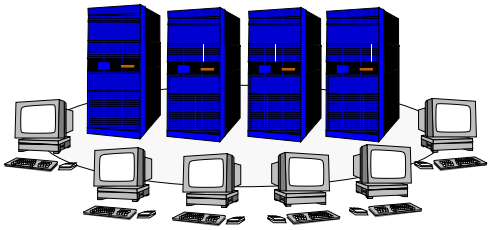
[1] As more analytically expressed by Metcalfe's law, where the *interest* of a communication system grows like $O(n^2)$ for n participants, or Reed's law, where the *utility* of the system increases like $O(2^n)$ (cf. Wikipedia, the free encyclopedia, `http://www.wikipedia.org`).



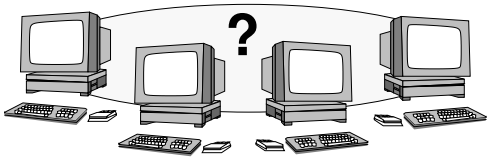
(a) A few users connect; ISPs like AOL, CompuServe, offer expensive services.



(b) Number of users increases; service providing in the Internet becomes a business for third parties.



(c) Affordable broadband, cheap powerful equipment become available; users are able to provide services themselves.



(d) What will the future be like ?

Figure 1.1: The evolution of the users and service providers in the Internet.

1.2 P2P Systems

P2P systems build on many concepts from the area of distributed systems: Algorithms for redundancy, load balancing, overlay networks or resilience are used as building blocks of these systems. But in what sense are P2P systems new? What does make them different enough so as to treat them separately?

Systems behaving in a Peer-to-peer fashion pre-existed the applications we call Peer-to-peer today. In networks like Fidonet or UUCPNet each computer could connect to others through telephone lines in order to transfer files. The Internet itself started as a network of point-to-point links on top of telephone lines. In the very beginning, all computers in the Internet were *peers*, because no Domain Name System (DNS) [23] existed and a host had to know another in order to establish a communication with the latter. Nowadays, many protocols work in a P2P fashion, at least between servers (NNTP, SMTP). Why do we talk this much [24] about P2P these days ? Because the kind of systems we consider in this thesis could not have been imagined when that primitive “Peer-to-peer” Internet existed. Only today we can start to build systems of millions of interconnected hosts. P2P systems are a consequence of the development of the Internet. We propose our own definition, as follows:

P2P systems are massively deployed distributed applications supported by the resources provided by the users on the edges and interconnected via the Internet.

It is only nowadays that such a system is feasible. As such, Peer-to-peer systems present their own characteristics:

- They are composed of a very large number of hosts, and thus their properties must scale gracefully with an increasing number of participants. Millions of users can be considered a normal scenario [25, 26, 27].
- The computing infrastructure is provided by the users' machines, which can be powerful but often unreliable. Commodity hardware is assumed for all the participating machines. Any given machine (peer) can fail [28] or disconnect [26] at any point in time.
- We consider justified to require a minimum bandwidth for some P2P applications to work. However, for most or all peers, bandwidth should be treated in general as a scarce resource [25, 26]. Thus a sense of economy must be applied to all communication not involving the actual exchange of information among users.
- Users are rational. They know that they must collaborate in order to benefit from the system. However, most users will not offer resources without an interest motivating them. In real life P2P networks, a few peers offer resources just for the sake of it, while others may not collaborate at all. In the middle of this range of behaviors we find normal users, which cooperatively build the common infrastructure, motivated by their own interest. An example of *mean*[2] behavior in file-sharing networks are *free riders* [29]. Free riders do not share any files, but they download content from other peers. Other mean peers make part of the P2P system only during the time needed to satisfy their interest, disconnecting afterwards. These and other types of mean peers, while connected, behave like any other normal peer, only that they do not have resources to offer. Malicious peers connecting to the P2P network and refusing to properly handle the protocols (or outrightly breaking them) are a security risk, because they intentionally disrupt the system with the only purpose of sabotage. Mean peers are not such a danger. Instead, they are one example of the differences between a P2P system and a traditional distributed system: the latter may be a dedicated infrastructure, but the former is rational. This cannot be ignored, and P2P systems may implement mechanisms to enforce collaboration [30]. Ultimately, a P2P system containing exclusively mean peers ceases to exist because no peer can obtain anything valuable from the others.

In general, a P2P system is (roughly) composed of a substrate to allow communication among peers, the algorithms to locate resources, and an application running on top of the distributed environment. We now define the key elements in a P2P network.

[2]We use the term *mean* to emphasize the fact that the peer finds valuable the very fact of not collaborating, that is, not letting others use its resources. A *selfish* peer collaborates indeed, motivated by its own interest, because the collaboration of all peers is needed to obtain a benefit from the network.

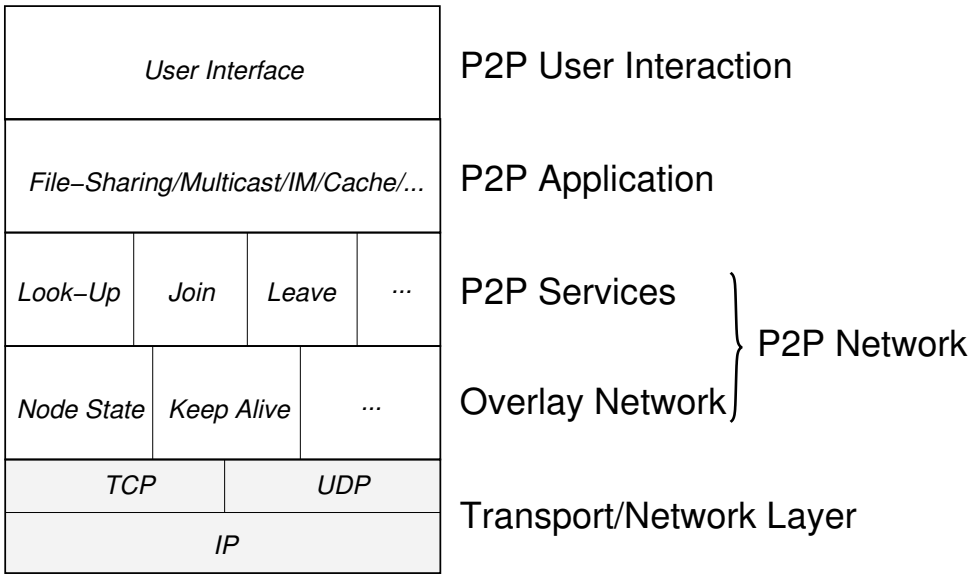


Figure 1.2: The Peer-to-peer Protocol Stack

In Figure 1.2 we see a detailed P2P protocol stack. Note that all the P2P layers are situated at the *application layer* of the OSI model.

What we call *P2P network* is formed by two closely related layers: the *Overlay Network* and the *P2P Services*.

- *Overlay network*: An Application-level routing scheme among the nodes of a distributed application. Packets from one node of the overlay network to another are driven on their way by the *underlay* routing at the network layer, through the OSI level 3 routers in the path from the first overlay node to the second. The overlay routing (i.e., deciding which is the next destination overlay node at each hop) is explicitly made at the application level at each overlay node through the path in the overlay network from the origin node to the final node. This is the reason why overlay networks are also called Application-level networks. A solid example is the Resilient Overlay Network [31] (RON): a group of multi-homed nodes assure continuous communication even in the presence of network failures, by routing packets through an application-level overlay path that avoids the failed network or link. Overlay networks are often represented by a directed graph (X,U), where X is the set of end nodes in the overlay and U is the set of application-level links among the nodes in X. The (X,U) graph is required to be connected.
- *P2P Services*: Functionality provided by the P2P network for the applications sitting on top. Each instance of a distributed application on each peer interacts with the other instances on other peers through the API offered by the P2P services. Examples of the functions in this API are joining a P2P network, or perform a look-up that finds the peer responsible for a given resource in the P2P network.

From now on, we may use the term *P2P network* to refer to the full Peer-to-peer system,

up to the P2P application. Although we do not agree with this identification of a system with one of its parts, such terminology is widely used. We believe this ambiguity will not mislead the reader, as the context will make clear the scope of the term (full system or just the P2P network). When we use the term *network* alone, we mean the underlying physical network up to layer 3 of the OSI model, IP in our case.

1.3 The Look-Up Algorithm in a P2P System

Resources in a P2P system are distributed over all peers. These resources must be first located in order access to them. A *key* identifies a resource in a P2P system. The more general forms found in the literature are a string of characters or a numerical value of fixed length. How a P2P system searches for resources using these keys, and how the keys relate to peers, define the kind of P2P system and will be explained below.

The systems defined in the previous section can be further divided into structured and unstructured P2P systems. The central difference between the two relies on the look-up method used to locate resources in the P2P network.

We define Look-Up Algorithm as the application-level distributed algorithm implemented at each node of an overlay network, allowing a query for a key k to reach the node responsible for that key.

Overlay and Layer-3 networks route *packets*. From here on, we refer to the pieces of data routed by P2P networks as *queries*. In both structured and unstructured P2P systems, each peer knows about a small (compared to the total peer population) subset of other peers referred to as *neighbors*. The population of peers and their connections to neighbors form the graph (X,U) that is the substrate of the P2P network. However, the neighbors of a peer (and thus the links in U) are chosen depending on the look-up algorithm used in the P2P network. So we can say that the look-up algorithm defines or specifies a P2P network, by determining the (X,U) graph of the overlay network (see Figure 1.2). Of course, we may directly focus on the restrictions imposed to the neighbor selection (and not the fact that it is the look-up algorithm which ultimately resolves which are the neighbors) and say that the rules defining which are the neighbors of a peer specify the overlay network. However, the P2P Services (and thus the look-up) and the overlay network are usually two so interrelated layers that both approaches are equally valid.

The set of neighbors of a peer is usually referred to as *routing table*, because the next hop for a query is decided among the neighbors of the peer taking the routing decision. A very interesting paper [32], which studies the trade-off between network diameter and routing table size, shows the close relation between the look-up algorithm and the neighbors in an overlay network.

We now explore the differences between structured and unstructured P2P systems, concerning keys, peers and look-up algorithm, using two well-known examples: Gnutella [18] and Chord [33].

Unstructured P2P Systems:

- *Keys:* They typically consist of a string of characters, from a namespace of arbitrary length. Keys usually contain a description of the resource they identify.

  Example: in file-sharing P2P applications like Gnutella, each key is part of the name of a file, or of its metadata such as ID3 tags in MPEG-1 Layer III (MP3) files.

- *Peers:* Peers are not identified by a proper name. Peers usually host only resources for which the user has a explicit interest. The neighbors of a peer are chosen at random from the total set of peers, although optimizations can be introduced to improve look-up performance, such as connecting to peers close in the physical network, or to peers that provide relevant information [34]. This last criterion leads to the formation of power-law networks [35], where most peers are connected to the small subset of the same peers. We say that these few peers have a *high in-degree*, that is, many links in (X,U) are directed towards them. Conversely, we say that those few peers have a *high fan-out*, that is, they serve content to many other peers.

  Example: In [35], the authors suggest choosing as neighbors the peers with most connections to other peers for improved search performance. The "small world" phenomenon [36] shows how surprisingly efficient search in these network can be. There is, however, the risk of overloading the peers that get many connections, as this triggers more and more connections from the other peers towards them, creating an avalanche effect.

- *Look-Up Algorithm:* When a peer searches for a given resource, it issues a query containing information that the key identifying the resource must match. The query is sent to a number of (possibly all) neighbors, which search for a matching key among the resources they are hosting. Those peers further forward the original query to their own neighbors, so that the search reaches eventually all peers. Matching keys are returned to the originator of the query as a result of the look-up. This peer may get overwhelmed by a gigantic set of results if the scope of the search or the results set size is not limited.

  Example: In Gnutella, a peer receiving a query searches among the files it stores, returns matching filenames to the peer the query came from and forwards the query to its own neighbors. The process [37] ends after a given number of forwarding steps, to avoid flooding the network. KaZaA works similarly, but introduces a special kind of peers, the superpeers, which are aware of the content of other peers and extend the role of the regular peers answering on their behalf.

Structured P2P Systems:

- *Keys:* In structured P2P systems, keys are fixed-length bit strings. That is, they are numerical identifiers from a finite set. There does not need to be an explicit relation between a resource and the key identifying it. An easy method to obtain keys for resources is through a hash function: the result of applying a hash function on a resource or some metadata related to that resource is the key. Note that in this case we introduce an explicit relation between the key and the resource.

  Example: In Chord, a key is a 160-bit string. Keys are obtained as the output of the SHA-1 [38] secure hash function.

- *Peers:* Peers are uniquely identified by a numeric ID from the same space as the keys. We abuse the notation, denoting a peer and its ID by the same symbol, usually p. The neighbors of a peer are precisely defined by the look-up algorithm. A peer is responsible for the resources that according to the look-up algorithm are mapped to said peer. A peer p is responsible for a key k if p is the peer with the closest ID to key k in the P2P network, according to some metric.

  Example: In Chord, for an identifier space I of m-bit strings, a peer of ID $p \in I$ has a set of neighbors consisting of approximately the peers with ID $q_i = \forall_{0 \leq i < m}(p + 2^i) \mod 2^m$. Peer p is responsible for all keys k with $q < k \leq p$, where q the is peer with the highest ID value smaller than p.

- *Look-Up Algorithm:* When a query for a key k is issued at peer p, p sends the query to the neighbor peer with the ID closest to k, according to some metric. Each peer receiving the query performs the same operation, until eventually the peer responsible for k is reached. The forwarding of the query can be done recursively, where each peer takes the routing decision and forwards the query, or iteratively, where at each step of the look-up, the ID of the next destination peer is returned to the peer originating the query and it is the latter who performs the forwarding.

  Example: Each peer p in Chord forwards a query for key k to the peer q in p's routing table with the highest ID, provided $k < q$.

Structured P2P systems are usually considered as Distributed Hash Tables (DHT). More precisely, it is the look-up algorithm of the structured P2P systems that is identified with this data structure. Indeed, a Hash Table [39] returns the bucket (entry) in a table corresponding to a given key. In this sense, the P2P look-up algorithm works similarly, returning a peer (location) in the system for a given key. Thus we define:

A Distributed Hash Table (DHT) is a look-up algorithm in a structured P2P system that uniquely maps a key k to a single peer p, and defines an overlay network substrate (X,U) through which queries for k are routed towards p.

The look-up algorithms offering DHT functionality differ from each other in the way a query for a key is routed to find the peer responsible for that key. We consider the identification

of structured P2P systems with DHTs valid in the cases where it is clear that the P2P system provides just a look-up functionality (that is, when we consider a P2P system consisting of a P2P network exclusively). In the case of a more general purpose P2P system, we prefer the term *P2P Look-Up Service* (as used in Figure 1.2) to refer to the implementation of the DHT used as look-up algorithm. The corresponding system can be named DHT-based P2P system for short.

The unstructured P2P systems have been presented here for completeness and to provide a proper background, but will not be the object of further study in this thesis. We will refer to them if needed for clarification or analogy.

We focus our work on the structured P2P systems from here on. Examples of structured P2P networks are Chord [33], CAN [40], Pastry [41], Tapestry [42], and P-Grid [43].

1.4 Topics and Motivations of the Thesis

In this section we present the main topics developed in this thesis. We will provide specific related work and full references for each subject in its corresponding chapter.

1.4.1 Organizing Peers for P2P Network Optimization

Most P2P systems are designed as flat organizations of identical peers. This means that the overlay layout looks exactly the same from any peer taken at random, and that each peer behaves in the same manner. Such system design leads to beautiful, symmetric designs, featuring nice properties, especially concerning their theoretical scalability.

It is, however, a fact that the physical network underlying those P2P systems is not equally perceived by all machines. The properties of the network change with the kind of connection used by each machine, its location, and the policies ruling connectivity between different regions of the Internet. But in the absence of some awareness of these facts, the P2P system cannot adapt to the surrounding environment.

The physical existence of the network implies a spatial distribution of its infrastructure and physical characteristics (e.g., latency) that the P2P system is constrained to deal with. Organizing peers so that the most efficient communication paths (e.g., whenever possible, contact peers in the same LAN) are used most of the time is a first optimization of a P2P system.

The machines forming the distributed system are those of the users and thus present very diverse characteristics. The capabilities, CPU and bandwidth, availability and reliability of a machine depend on both hardware and software, but also on the user. Peers hosted by these machines are logically very different from each other. For a given figure of merit of the system,

like performance or reliability, the characteristics of some peers are more suited than others. A flat system places all peers at the same level, and every peer can be potentially affected by the lack of performance of another. Proper organization can achieve further optimization by letting more powerful peers do tasks that are more critical to achieve a given performance goal.

We have chosen the hierarchy as the method of organization for the peers in the P2P system. Hierarchies adapt well to the infrastructure of the Internet. Hierarchies present good scalability properties, because their members form a tree: the number of hops between any two nodes in the hierarchy is proportional to the logarithm of the number of nodes of the structure, and each node needs to know only a reduced set of neighbors. However, hierarchies present single points of failure at each level: the failure of a peer at one level of the hierarchy disconnects the peers below from the rest of the structure. P2P systems are formed by a large number of machines, which allow the construction of redundancy at each level of the hierarchy. This approach is similar to *fat trees* [44], which double the number of parents of a node the higher it is in the hierarchy (resulting in a "tree" with multiple roots). This way we reinforce the collaborative aspect inherent to a P2P system even if we introduce separate scopes for the actions of peers, depending on their level in the hierarchy.

We further explore hierarchical DHTs in Chapter 2.

1.4.2 Overlay Network and Physical Network

In principle, a proper protocol stack (Figure 1.2) provides isolation between layers that communicate exclusively through their interfaces. However, the overall performance of such a stack may be very poor even if each layer taken separately correctly accomplishes its task. The overall performance of the implementation of a given protocol stack can be improved if the different layers *share information* about their characteristics so that their behavior is *optimized* by better adapting to the environment they work in. We say then that one layer is *aware* of another. This awareness can be obtained at execution time, or be introduced at the design of the stack.

- When obtained during execution time, a layer learns about the characteristics of another layer below. This learning is usually done by probing some magnitude of interest at the inferior layer. The probes permit to obtain a representation of the real conditions on the probed layer. Depending on the values obtained with the probes, some behavior is decided to achieve improved performance. Periodical probing allows for adaptation to changing conditions, choosing a new behavior accordingly.
- If we know the properties of the environment where the implementation of the stack is going to work, we can use this knowledge in the design of the layers. We make one layer aware of another by making the first assume the second's inner workings. By making a layer aware of another the first can be optimized accordingly to obtain some predefined goal. Because we hardwire these optimizations in the stack design, they must correspond to some invariant in the working environment.

The first approach is more flexible, because a layer implementation can be substituted by another and the stack still works. The aware layer would only need to continue probing in order to infer the characteristics of the new layer. For example, in Figure 1.2, the overlay network would change the links in its (X,U) substrate based on the measurements taken at the Transport/Network layer. This method, however, can only optimize the behavior of a layer on the basis of the *representation* it makes itself of another, and can only be accurate to some extent. Moreover, the need for probing introduces extra traffic in an environment where bandwidth (and also available memory or processing power) may be a scarce resource.

The second method can push the optimization of the stack further, because it knows the characteristics of the different layers. It presents the inconvenience of being valid only for the scenario the design is optimized for: A change in a layer breaks the assumptions made by another layer, invalidating the awareness the latter had of the former. For instance, an overlay network designed so that it best benefits from the characteristics of a Token-Ring network, may not be simultaneously the best choice for an Internet-wide deployment.

We propose a topology-aware DHT designed with the structure of the IPv4 Internet in mind. Peers are organized in groups that correspond to IP network prefixes, which are further organized in a hierarchy, imitating hierarchical IP prefix aggregation in BGP[3] [45] routers. Obviously, this approach is very well suited (as we show later) to the current Internet, but not to other scenarios. Hierarchical prefix organization is further enforced in IPv6, which makes us confident on the validity of our design in the long term.

We further elaborate on this subject in Chapter 3.

1.4.3 Optimization Through Topology-awareness

It has usually been the case that an application-level network considered the physical network as a transparent layer, transmitting data from one point of the overlay to another. This would indeed be a common scenario if network resources were abundant. This may be the case in a small LAN where all machines can be assumed to be connected in a mesh with sufficient bandwidth, or when the generated traffic is negligible compared to bandwidth resources available. Many networks, however, have to deal with economic aspects, and must thus be engineered to optimize their utilization.

Thus, as illustrated in Figure 1.3, a population of n users, each of them connected to a link of bandwidth b, is in general given a total bandwidth C to serve them all, with $C < b \cdot n$.

At the time when the number of simultaneous users n was much lower than today, and most of them connected to the Internet through links of low bandwidth b, not a lot of bandwidth C was required. Users connected just to query some services provided by well-connected servers.

[3] For a very brief introduction to BGP, we refer the reader to Chapter 3.

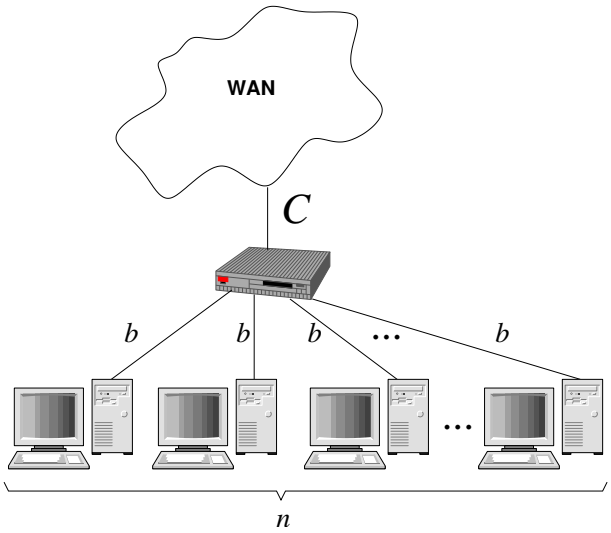


Figure 1.3: Concentration of resources (bandwidth) in the Internet.

With n increasing massively and b growing to the Megabit per second range (DSL) for individual users or up to a hundred Megabits per second for corporate users, C is being upgraded fast. But users do not connect exclusively from time to time anymore, thanks to flat-rate offers from ISPs. Moreover, as users are now able to offer services to others collaborating in P2P networks, they can be generating traffic permanently. Will the Internet become a huge LAN, a full mesh with $C = b \cdot n$? Perhaps one day, but in the meantime optimizations are required for better resource utilization. A network hosting a number of peers in a P2P system may find its outbound link saturated by *identical* data flowing to and from its local peers, unless those peers are somehow *aware* of the fact that they are sharing the same network. If peers in the same network become topology-aware, they can cooperate so that if multiple copies of some data are required, just one copy is transmitted over outgoing link (like HTTP proxies do today for Web traffic). Furthermore, data can be obtained from peers in other closeby networks through low-latency and often higher bandwidth links, to improve response time. Then, copies can be distributed inside the network.

We use our topology-aware DHT to distribute content over a P2P network using application-level multicast. Our algorithms take advantage of the topology-awareness of the P2P network so that each peer obtains the data from closeby peers, while avoiding bottlenecks at the access links of networks. These features are obtained with no need for probing the network, because the topology-awareness is introduced in the design of the underlying DHT. All the available bandwidth can be dedicated to data transmission.

This topic is treated mainly in Chapters 4 and 5.

1.4.4 Search in Structured P2P Systems

Although DHT look-up services are very efficient at finding the location of keys in a P2P network, keys themselves are not the matter of interest to the users of P2P systems, they just serve as pointers to content or other resources. Moreover, keys are arbitrary bit strings that mean nothing to a user, and thus cannot be used directly to search data. In order to provide utility to the users of a P2P system, the ability of the DHT to find keys must be made available through a more intuitive interface.

A first naive approach would be to have a search service with an infrastructure independent of the P2P network (e.g., a web page). This search service would provide a key or a collection of them in response to a user query (e.g., some keywords describing a resource). However, if this service was attacked or simply unavailable because of a technical problem, the whole P2P system would remain useless, because of the impossibility of finding anything inside.

The search mechanism should thus be implemented as a P2P system, in tight relation with the content storage P2P system. Each peer could implement a small search engine, and upon arrival of a query, a peer would search among the contents it is storing, reply with possible answers if anything is found, then forward the query to other peers. Such is the approach of Gnutella and other unstructured P2P systems. However in DHT-based P2P storage systems peers do not normally know what they are storing, and even in that case, this approach, at least in it most straightforward form, does not take advantage of the efficient key look-up.

We propose to index the resources stored in a P2P storage system, and then to store the indices like any other content. For each resource, a list of descriptors is generated. These descriptors are used as indices, following a simple rule: One index points to the indices that provide more specific information about a resource. We obtain a key from each index. The content found upon look-up of a key is the set of indices pointed by the index corresponding to the key. The index describing completely a resource points to the resource itself. Thus the search through the index is performed as a series of look-ups of keys in the DHT. This approach achieves both goals, integration with the P2P network and taking real benefit from DHT's features.

We further develop the indexing technique in Chapter 6.

1.5 Topics not Covered in This Thesis

P2P systems are a recent topic and many new issues arise, as new possibilities are discovered. There are at least two topics that we consider important in regard of P2P networks, but not treat in our thesis:

- **Security:** We do not cover basic security services in Peer-to-peer networks. The techniques that allow avoiding the pernicious effects of a badly behaving (possibly colluding) group of peers are out of the scope of this work. However, where possible, we succinctly address some security issues by suggesting how the P2P system may detect uncooperative peers.

  The absence of a centralized point of failure in a P2P network makes impossible a centralized *trusted third party* (e.g., a certification authority), which is the base for classic authentication, non-repudiation and confidentiality in the Internet [46, 47]. There are efforts to distribute the certification authority among a number of machines through threshold cryptography [48]. Threshold cryptography allows for the cooperative verification of a signature by different collaborative entities, making the collusion among peers more difficult to some extent. EigenTrust [49] is a distributed approach for peer reputation in a given P2P system. This method tries to avoid collusion among a group of peers providing faked reputation for each other. The problem posed by arbitrary peer identity is treated in the Sybil Attack [50]. The main issue is that a peer can fake an unlimited number of identities and use them to insert itself at multiple locations of the P2P network. This possibility can be misused to alter the normal workings of a P2P network, although it can also serve legitimate purposes (e.g., to balance load in the network through *virtual peers* [33]). Secure services through overlay networks is the topic of [51]. Finally, censorship resistant P2P content addressable networks are explained in [52]. The authors present a scalable approach allowing all but an arbitrary small fraction of the peers in a P2P network access to all but a small fraction of the content in that P2P network, after an arbitrary number of peers have been removed by an adversary.

- **Economics:** The massive spread and ease of utilization of many file-sharing P2P networks have severely affected certain industries, which have seen their copyrighted material shared and downloaded for free. However, others see in P2P systems a new way of distributing content and the birth of a new economy.

  Topics treated traditionally in Economy like pricing [53] or incentives [54] are related to security. Indeed, mechanisms to assure confidentiality, trust [55, 30] or reputation [56, 57] are necessary for an economic system to work properly. Some approaches attempt to imitate the economic exchanges in our societies to develop optimal allocation of resources in P2P networks [58].

1.6 Organization of This Thesis

The improvement in technology has given the users of the Internet network bandwidth and computers with capabilities no one could dream of few years ago. Users have the possibility to offer themselves new or modified services not available by traditional means. However, the infrastructure needed for efficient and economic Internet services can only be obtained by gathering the resources distributed over the edges of the Internet: such is the role of P2P systems.

Our aim is to present a system that provides a content distribution and location infrastructure to the peers involved. Such a system attempts to efficiently use resources and optimize performance by the usage of a hierarchical DHT with a topology-aware design as look-up service.

In Chapter 2 we introduce and develop the hierarchical DHT concept. We explain their properties, how they relate to the current Internet infrastructure, and why they are of interest to us. We study the impact of the introduction of a hierarchical design into a well-known DHT.

Chapter 3 presents a specific hierarchical DHT, named TOPLUS, which stands for TOpology-aware Look-Up Service. TOPLUS' main characteristic is the topology-awareness introduced in its design. As a DHT, TOPLUS allows for contacting the peer responsible for a key in a time equivalent or at most proportional to that of the IP routing latency from the querying peer to the destination peer. We specify its inner workings and main properties, and we present experimental results on its performance as a look-up service.

Chapters 4 and 5 present MULTI+, the content distribution network built on top of TOPLUS. MULTI+ works as an application-level multicast service. By using the properties of TOPLUS, MULTI+ organizes peers so that they obtain data from other peers close by while avoiding bottlenecks in the network. Chapter 4 explains the MULTI+ design and properties, and shows experimental results on the evaluation of its capabilities. In Chapter 5 we study the effect of the massive failure of peers on the performance of the multicast trees built with MULTI+. We propose a simplified approach for a possible implementation. This division in two chapters may appear arbitrary, but its only aim is to make the reader's task more comfortable.

In Chapter 6 we propose a model for data location on structured P2P networks. Our model is based on the indexing of data previous to storage in the P2P system. Storing indices as common data permits a user to obtain an index just by using the DHT look-up, performing the search without extra infrastructure added to the P2P system. The methods explained can be applied to any DHT look-up service.

Chapter 7 contains a resume of the presented work, and concludes this thesis by proposing new directions for research.

Chapter 2

Hierarchical Peer-to-peer Systems

There never were in the world two opinions alike, no more than two hairs or two grains; the most universal quality is diversity.

Michel de Montaigne. Essays

Structured Peer-to-peer (P2P) look-up services organize peers into a flat overlay network and offer Distributed Hash Table (DHT) functionality. Data is associated with keys and each peer is responsible for a subset of the keys. In hierarchical DHTs, peers are organized into groups, and each group has its autonomous intra-group overlay network and look-up service. Groups are organized in a top-level overlay network. To find a peer that is responsible for a key, the top-level overlay first determines the group responsible for the key; the responsible group then uses its intra-group overlay to determine the specific peer that is responsible for the key. We provide a general framework for hierarchical DHT look-up services and we propose a scalable hierarchical overlay management. Then we consider the case of a two-tier hierarchy using Chord for the top level. Our analysis shows that by designating the most reliable peers in the groups as superpeers, the hierarchical design can significantly reduce the expected number of hops in Chord.

2.1 Introduction

Peer-to-peer (P2P) systems are gaining increased popularity, as they make it possible to harness the computing power and resources of large populations of networked computers in a cost-effective manner. A central problem of P2P system is to assign and locate resources among peers. This task is achieved by a P2P *look-up service* (see Figure 1.2).

Several important proposals have been recently put forth for implementing distributed P2P look-up services, including Chord [33], CAN [40], Pastry [41] and Tapestry [42]. In these look-up services, each key for a data item is assigned to the live peer whose ID is "closest" to the key (according to some metric). The look-up service essentially performs the basic function of determining the peer that is responsible for a given key. The look-up service is implemented by organizing the peers in a structured overlay network, and routing a message through the overlay to the responsible peer. The efficiency of a look-up service is generally measured as a function of the number of peer hops needed to route a message to the responsible peer, as well as the size of the routing table maintained by each peer. For example, Chord requires $O(\log N)$ peer hops and $O(\log N)$ routing table entries when there are N peers in the overlay. Implementations of the distributed look-up service are often referred to as *Distributed Hash Tables (DHTs)*: terminology has been discussed in Section 1.2.

DHTs are the central components of a wide range of new distributed applications, including distributed persistent file storage [59, 60], Web caching [61], multicast [62, 63], or computational grids [64]. DHTs generally provide improvement to an application's resilience to faults and attacks.

Chord, CAN, Pastry and Tapestry are all flat DHT designs without hierarchical routing. Each peer is indistinguishable from another in the sense that all peers use the same rules for determining the routes for look-up messages. This approach is strikingly different from routing in the Internet, which uses hierarchical routing. Specifically, in the Internet, routers are grouped into Autonomous Systems (ASes). Within an AS, all routers run the same intra-AS routing protocol (e.g., OSPF). Special gateway routers in the various ASes run an inter-AS routing protocol (BGP) that determines the path among the ASes. Hierarchical routing in the Internet offers several benefits over non-hierarchical routing, including scalability and administrative autonomy (e.g., at the level of a university, a corporate campus, or even the coverage area of a base station in a mobile network).

In this chapter we explore hierarchical DHTs. Inspired by hierarchical routing in the Internet, we examine two-tier DHTs in which

1. peers are organized in disjoint groups, and
2. look-up messages are first routed to the destination group using an inter-group overlay, and then routed to the destination peer using an intra-group overlay.

We will argue that hierarchical DHTs have a number of advantages, including:

- They significantly reduce the average number of peer hops in a look-up, particularly when peers have heterogeneous availabilities.
- They significantly reduce the look-up latency when the peers in the same group are topologically close and cooperative caching is used within the groups.

- They facilitate the large-scale deployment of a P2P look-up service by providing administrative autonomy to participating organizations. In particular, in the hierarchical framework that we present, each participating organization (e.g., institutions and ISPs) can choose its own look-up protocol (e.g., Chord, CAN, Pastry, Tapestry).

We present a general framework for hierarchical DHTs. In the framework, each group maintains its own overlay network and uses its own intra-group look-up service. A top-level overlay is also defined among the groups. Within each group, a subset of peers are labeled as "superpeers". Superpeers, which are analogous to gateway routers in hierarchical IP networks, are used by the top-level overlay to route messages among groups. We consider designs for which peers in the same group are locally close. We describe a cooperative caching scheme that can significantly reduce average data transfer delays. Finally, we also provide a scalable algorithm for assigning peers to groups, identifying superpeers, and maintaining the overlays.

After presenting the general framework, we explore in detail a particular instantiation in which Chord is used for the top-level overlay. Thus, in this instantiation, Chord is analogous to BGP in Internet routing, and the intra-group look-up services are analogous to intra-AS routing protocols. Using a novel analytical model, we analyze the expected number of peer hops that are required for a look-up in the hierarchical Chord instantiation. Our model explicitly captures inaccuracies in the routing tables due to peer failures.

The chapter is organized as follows: We first discuss related work in Section 2.2. We then present the general framework for hierarchical DHTs in Section 2.3. We discuss the particular case of a two-tier Chord instantiation in Section 2.4, and we quantify the improvement of look-up latency due to the hierarchical organization of the peers.

2.2 Related Work

As we have said in Chapter 1, P2P networks can be classified as being either unstructured or structured. Chord [33], CAN [40], Pastry [41], Tapestry [42], and P-Grid [43], which construct highly structured overlays and use hashing for targeted data placement, are examples of structured P2P networks. These P2P networks are all flat designs (P-Grid is based on a virtual distributed search tree, but peer nodes are located at the leaves only and the tree is used solely for routing purposes). Gnutella [18] and KaZaA [20], whose overlays grow organically and use random data placement, are examples of unstructured P2P networks.

The authors of [65] explore using landmark nodes to bin peers into groups. The basic idea is for each peer to measure its round-trip time (RTT) to M landmarks, order the resulting RTTs, and then assign itself to one of $M!$ groups. The authors then apply this binning technique to CAN, to construct a locality-aware overlay. In this binning-CAN scheme, the node ID space is partitioned into $M!$ equal-size portions, one portion corresponding to each group. When a peer wants to join the overlay, it pings the landmarks to determine the group, and hence the

portion of the ID space, to which it belongs; the peer then gets assigned a node ID, uniformly chosen from that portion of the node ID space. This implies that during a look-up, typically short topological hops are taken while the look-up message travels through a group; and then longer topological jumps are taken when the message reaches the boundary of a group. Our hierarchical DHT schemes bear little resemblance to the scheme in [65]. Although in [65] the peers are organized in groups according to locality, the look-up algorithm applies only to CAN, does not use superpeers, and is not a multi-level hierarchical algorithm.

Our approach has been influenced by KaZaA, a very successful unstructured P2P file sharing service. (Today, KaZaA has typically several million participating peers at the same time). KaZaA designates the more available and powerful peers as *supernodes*. In KaZaA, when a new peer wants to join, it bins itself with the existing supernodes, and establishes an overlay connection with the supernode that has the shortest RTT. The supernodes are connected through a top-level overlay network, using a proprietary design. A similar architecture has been proposed in CAP [66], a two-tier unstructured P2P network that focuses on scalability and stability. Analyzing the properties of Gnutella networks [67, 68], it is clear that peers tend to organize themselves around the more stable peers in the network. A new proposed standard for the Gnutella protocol also abandons pure flat structures [69]. Our design is a blend of the supernode/hierarchy/heterogeneity of KaZaA with the look-up services in the structured DHTs.

Brocade [70] proposes to organize the peers in a two-level overlay. All peers form a *single* overlay O_L. Geographically close peers are then grouped together and get assigned a representative called "supernode". Supernodes are typically well connected and situated near network access points. The supernodes form another overlay O_H, and each of them must somehow announce which peers are reachable through him. Brocade is not truly hierarchical since *all* peers are part of O_L, which prevents it from reaping the benefits of hierarchically organized overlays discussed in section 2.3.1.

Finally, the authors of [71] present a topology-aware version of Pastry [41]. At each hop, Pastry presents multiple equivalent choices to route a request. By choosing the closest (smallest network delay) peer at each hop, they try to minimize network delay. However, at each step the possibilities decrease exponentially, so delay is mainly determined by the last hop, usually the longest. Our approach is somewhat the opposite, as we propose large hops first to get to a group, and then shorter "local" hops inside the group. Note that our architecture leads to a more natural caching scheme, as shown later in section 2.3.4.

2.3 Hierarchical Framework

We begin by presenting a general framework for a hierarchical DHT. Recall (see Section 1.3) that we use DHT as an abbreviation for a structured P2P network with a DHT-based look-up service. We accept this identification because the P2P systems of concern in this chapter offer exclusively look-up functionality. Although we focus on a two-tier hierarchy, the framework

can be extended to a general n-tier hierarchy in a straightforward manner.

Let N denote the set of peers participating in the system. Each peer has a peer ID. Each peer also has an IP address, which may change whenever it re-connects to the system. The peers are interconnected through a network of links and switching equipment (routers, bridges, etc). The peers send look-up query messages to each other using a hierarchical overlay network, as described below.

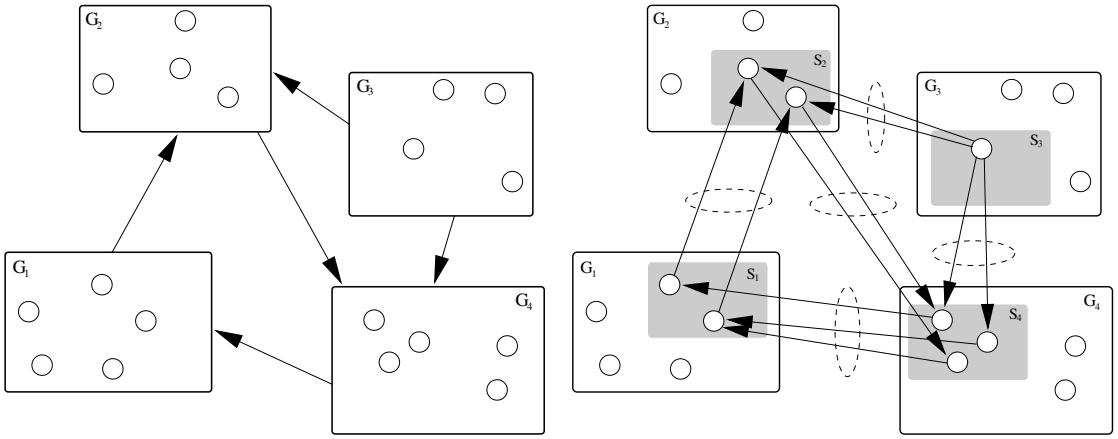
The peers are organized into groups. We will discuss how groups are created and managed and how peers are assigned to groups in Section 2.3.3. The groups may or may not be such that the peers in the same group are topologically close to each other, depending on the application needs. Each group has a unique group ID. Let I be the number of groups, G_i the peers in group i, and abusing the notation, let it also be the ID for group i.

The groups are organized into a *top-level overlay network* defined by a directed graph (X,U), where $X = \{G_1,\ldots,G_I\}$ is the set of all the groups and U is a given set of edges between the nodes (that is, groups) in X. The graph (X,U) is required to be connected, that is, between any two nodes G and G' in X there is a directed path from G to G' that uses the edges in U. It is important to note that this overlay network defines directed edges among groups and not among specific peers in the groups.

Each group is required to have one or more *superpeers*. Superpeers, as we will discuss below, have special characteristics and responsibilities. Let $S_i \subseteq G_i$ be the set of superpeers in group i. Our architecture allows for $S_i = G_i$ for all $i = 1,\ldots,I$, in which case all peers are superpeers (and distinction between regular peers and superpeers becomes superfluous). We refer to architectures for which all peers are superpeers as the *symmetric design*. Our architecture also allows $|S_i| = 1$ for all $i = 1,\ldots,I$, in which case each group has exactly one superpeer. Let $R_i = G_i \setminus S_i$ be the set of all "regular peers" in group G_i. For non-symmetric designs ($S_i \neq G_i$), an attempt is made to designate the more powerful peers as superpeers. By "more powerful," we primarily mean the peers that are up and connected the most. But as secondary criteria, superpeers will be the peers that have network connection bandwidth and/or high CPU power.

The superpeers are gateways between the groups: they are used for inter-group query propagation. To this end, we require that if s_i is a superpeer in G_i, and (G_i,G_j) is an edge in the top-level overlay network (X,U), then s_i knows the name and the current IP address of at least one superpeer $s_j \in S_j$. With this knowledge, s_i can send query messages to s_j. On the other hand, if p is a regular peer, then p must first send intra-group query messages to a superpeer in its group, which can then forward the query message to another group. Regular peers must thus know the name and IP address of the superpeers in their group. Figure 2.1(a) shows a top-level overlay network. Figure 2.1(b) shows possible communication relationships between the corresponding superpeers. Figure 2.2 shows an example for which there is one superpeer in each group and the top-level overlay network is a ring.

Within each group there is also an overlay network that is used for query communication among the peers in the group. Each of the groups operates autonomously from the other groups.



(a) Between groups. (b) Between superpeers in neighboring groups.

Figure 2.1: Communication relationships in the overlay network.

For example, some groups can use Chord, others CAN, others Pastry, and yet others Tapestry. The ID space of peers may be shared among all groups or not. The DHT inside a group avoids by construction the presence of two peers with the same ID. If two peers in different groups have the same ID at their corresponding DHTs, globally unique identifiers can be easily obtained by the concatenation of the corresponding group ID and peer ID. This is valid for any number of levels, because uniqueness of the group ID at each level is assured by the DHT at that level.

2.3.1 Hierarchical Look-Up Service

Let us first consider a two-level look-up service, where the top level manages peer groups and the bottom level peer nodes. Given a key k, we say that group G_j is responsible for k if G_j is the "closest" group ID to k among all the groups. Here "closest" is defined by the specific top-level look-up service (e.g., Chord, CAN, Pastry, or Tapestry).

The implementation of the look-up service exploits the hierarchical architecture: first, the look-up service finds the group that is responsible for the key; then it finds the peer within that group that is responsible for the key. Specifically, our two-tier DHT operates as follows. Suppose a peer $p_i \in G_i$ wants to determine the peer that is responsible for a key k.

1. Using the overlay network in group i, peer p_i sends a query message to one of the superpeers in S_i.

2. Once the query reaches a superpeer, the top-level look-up service routes the query through (X,U) to the group G_j that is responsible for the key k. During this phase, the query only passes through superpeers, hopping from one group to the next. A superpeer in one group

uses its knowledge of the IP addresses of superpeers in the subsequent group along the route to forward the query message from group to group. Eventually, the query message arrives at some superpeer $s_j \in G_j$.

3. Using the overlay network in group j, the superpeer s_j routes the query to the peer $p_j \in G_j$ that is responsible for the key k.

4. Peer p_j sends a response back to querying peer p_i. Depending on the design, this response message can follow the reverse path of the path taken by the query message, or can be sent directly from peer p_j to peer p_i (ignoring the overlay networks), or further, be routed to the sender through the overlay using p_i as the key.

This approach can be generalized to an arbitrary number of levels. A request is first routed through the top-most overlay network to some superpeer at the next level below, which in turn routes the request through its "local" overlay network, and so on until the request finally reaches some peer node at the bottom-most level. In the rest of this chapter, we will focus on the case of a two-level look-up service.

The hierarchical architecture has several important advantages when compared to the flat overlay networks.

- *Exploiting heterogeneous peers:* By designating as superpeers the peers that are "up" the most, the top-level overlay network will be more stable than the corresponding flat overlay network (for which there is no hierarchy). This increased stability enables the look-up service to approach its theoretical optimal look-up hop performance (for example, on average $\log \frac{N}{2}$ for Chord, where N is the number of peers in the Chord overlay).

- *Transparency:* When a key is moved from one peer to another within a group, the search for the peer holding the key is completely transparent to the top-level algorithm. Similarly, if a group changes its intra-group look-up algorithm, the change is completely transparent to the other groups and to the top-level look-up algorithm. Also, the failure of a regular peer $r_i \in G_i$ (or the appearance of a new peer) will be *local* to G_i; routing tables in peers outside of G_i are not effected. As peers do in flat overlays by means of virtual nodes, more populated (powerful) groups can be assigned a larger fraction of the system load by using multiple entries in the top-level overlay for each of their superpeers.

- *Faster look-up time:* Because the number of groups will be typically orders of magnitude smaller than the total number of peers, queries travel over fewer hops. As we shall soon see, this property along with the enhanced stability of the top-level overlay can significantly reduce querying delays.

- *Fewer messages in the wide-area:* If the most stable peers form the top-level DHT, most overlay reconstruction messages happen inside groups, which gather peers that are topologically close. Fewer hops per look-up means also fewer messages exchanged for the same number of requests. Finally, as we shall see shortly, the hierarchical organization

of the peer groups is perfectly adapted to content caching, which can further reduce the number of messages that need to get out of the group.

2.3.2 Intra-Group Look-Up

The framework we just described is quite flexible and accommodates any one of a number of overlay structures and look-up services at any level in the hierarchy. At the intra-group level, the groups can use different overlays, which could all be different from the top-level overlay structure.

If a group has a small number of peers (say, in the tens), each peer in the group could track all the other peers in the group (their ids and IP addresses); the group could then use CARP [72] or consistent hashing [73] to assign and locate keys within the group. The number of steps to perform such an intra-group look-up in the destination group is $O(1)$, since each peer runs a local hash algorithm to determine the peer in the group responsible for a key (G_2 in Figure 2.2).

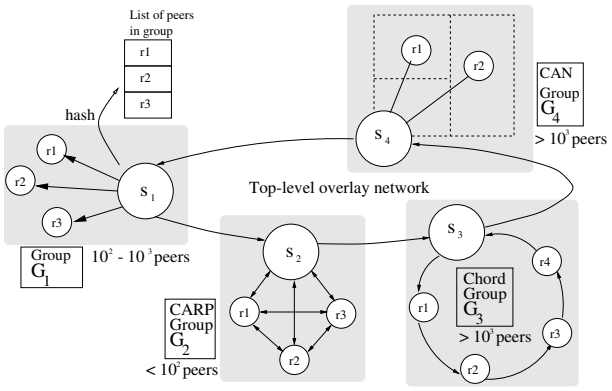


Figure 2.2: The case of a ring-like overlay network with a single superpeer per group. Intra-group look-up is implemented using different look-up services (CARP, Chord, CAN).

If the group is a little larger (say, in the hundreds), then the superpeers could track all the peers in the group. In this case, by forwarding a query to a local superpeer, a peer can do a local look-up in $O(1)$ steps (G_1 in Figure 2.2).

Finally, if the group is large (say, thousands of peers or more) then a DHT such as Chord, CAN, Pastry, or Tapestry can be used within the group (G_3 and G_4 in Figure 2.2). In this case, the number of steps in the local look-up is $O(\log |G_i|)$, where $|G_i|$ is the number of peers in the group.

2.3.3 Hierarchy and Group Management

We now briefly describe the protocols used to manage groups: consider peer p joining the hierarchical DHT. We assume that p is able to get the ID G of the group it belongs to. G may correspond to the name of p's ISP or university campus, or the hash of the domain name or the LAN IP network prefix, if locality of peers matters for the construction of groups. If it is not the case, G may be anything the peers of the group can agree on, like the topic of discussion in a forum. Thus, given this G, p first contacts and asks another peer p' already part of the P2P network to look up key G. Following the first step of the hierarchical look-up, p' locates and returns the IP address of the superpeer(s) of the responsible group. If the group ID of the returned superpeer(s) is precisely G, then p joins the group using the regular join mechanisms of the underlying intra-group DHT; additionally, p notifies the superpeer(s) of its CPU and bandwidth resources. If the group ID is not G, then a new group is created with ID G and p as only (super)peer.

In a network with m superpeers per group, the first m peers to join a group G become the superpeers of that group. Because superpeers are expected to be the most stable peers, we let superpeers monitor the peers that join a group and identify the ones that are highly available. Superpeers keep an ordered list of the superpeer candidates: the longer a peer has been connected and the higher its resources, the higher position in the candidate list the peer will be. A study on current unstructured P2P networks [74] shows that the longer a peer has been in the system, the longer that peer is likely to remain connected. A superpeer therefor needs to simply monitor *how long* each peer has been available to determine the order in its candidate list. This list is sent periodically to the regular peers of the group. When a superpeer fails or disconnects, the first regular peer in the list becomes superpeer and joins the top-level overlay. It informs all peers in its group, as well as the superpeers of the neighboring groups. Although large groups should be avoided, in those cases it may not be cost-effective to broadcast that information to all peers. Instead, peers can learn lazily about changes in the network (like new superpeers), for example by letting superpeers embed membership updates in the look-up responses. Epidemic algorithms [75, 76] can be used inside groups to propagate state updates.

Unavailability of peers due to some hardware or software problem should be less frequent than that due to user behavior (e.g., disconnection or careless termination). Still, see [28] for a study on the availability of the machines in a commercial environment. .

We are thus able to provide stability to the top-level overlay by using multiple superpeers, promoting the most stable peers as superpeers, and rapidly repairing the infrequent failures or departures of the superpeers.

2.3.4 Content Caching

In many P2P applications, once a peer p determines the peer p' that is responsible for a key, p then asks p' for the file associated with the key. This file might be a media file, a software package, a document, etc. If the path from p' to p traverses a congested or low-speed link, the file transfer delay will be long. In this section, we describe how hierarchical DHTs can be used to create groups of cooperative caches, which can significantly reduce file transfer delays.

In many hierarchical DHT setups, we expect the peers in a same group to be topologically close and to be interconnected by high-speed links. For example, a group might consist of peers in a corporate or university campus, with campus networks using 100 Mbps Ethernet. In such an environment, it is typically faster to retrieve files from other peers in the local group, rather than to retrieve the file from the global Internet. Also, by frequently confining file transfers to intra-group transfers, we reduce traffic loads on the access links between the groups and higher-tier ISPs.

Such hierarchical setups can be naturally extended to implement cooperative caching. Consider the following modification to the look-up algorithm. When a peer $p \in G_i$ wants to obtain the file associated with some key k, it first uses group G_i's intra-look-up algorithm to find the peer $p' \in G_i$ that would be responsible for k if G_i were the entire set of peers. If p' has a local copy of the file associated with k, it returns the file to p; otherwise, p' obtains the file (using the hierarchical DHT), caches a copy, and forwards the file to p. In this manner, files are cached in the groups where they have been previously requested (until they are replaced by a cache replacement algorithm, such as Least Recently Used – LRU). If a significant fraction of requests are for popular files, then a significant fraction of file transfers will be intra-group transfers. Thus, the hierarchical design can be used to create a P2P content distribution network. As a function of file sizes, cache sizes, request distributions and access bandwidth, one can use standard analytical techniques [77] to quantify the reduction in average file transfer time and load on access links.

2.4 Chord Instantiation

For the remainder of the chapter we focus on a specific top-level DHT, namely, Chord. In this context, we show that the two-tier design can significantly reduce look-up latency and file transfer latency.

2.4.1 Overview of Chord

In Chord, each peer and each key has a m-bit ID. IDs are ordered on a circle modulo 2^m. Key k is assigned to the first peer whose identifier is equal to or follows k in the identifier space. This

peer is called the successor of key k.

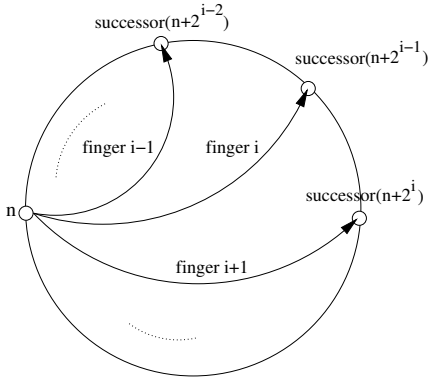


Figure 2.3: Fingers of a peer in a Chord ring.

Each peer tracks its successor and predecessor peer in the ring. In addition, each peer tracks m other peers, called *fingers*; specifically, a peer with ID p tracks all the successors of the IDs $p+2^{j-1}$ for each $j = 1,\ldots,m$ (note that p's first finger is in fact its successor). The successor, predecessor, and fingers make up the Chord routing table (see Figure 2.3).

During a look-up, a peer forwards a query to the finger with the largest ID that precedes the key value. The process is repeated from peer to peer until the peer preceding the key is reached, which is the "closest" peer to the key. When there are N peers, the number of hops needed to reach the destination is $O(\log N)$ [33].

2.4.2 Inter-Group Chord Ring

In the top-level overlay network, each "node" is actually a group of peers. This implies that the top-level look-up system must manage an overlay of groups, each of which is represented by a set of superpeers that may fail independently from the group they belong to. Chord requires some adaptations for being used in the top-level overlay network and managing groups instead of nodes. We will refer to the modified version of Chord as "top-level Chord".

Each node in top-level Chord has a predecessor and successor *vector* (instead of a pointer). The vectors hold the IP addresses of the superpeers of the predecessor and successor group in the ring, respectively. Each finger is also a vector, which holds the IP addresses of the superpeers of another group on the ring. The routing table of a top-level Chord group is shown in Figure 2.4.

The population of groups in the top-level overlay network is expected to be rather stable. However, individual superpeers may fail and disconnect the top-level Chord ring. When the identity of the superpeers S_i of a group G_i changes, the new superpeers eagerly update the vectors of the predecessor and successor groups. This guarantees that each group has an up-to-date view of its neighboring groups and that the ring is never disconnected.

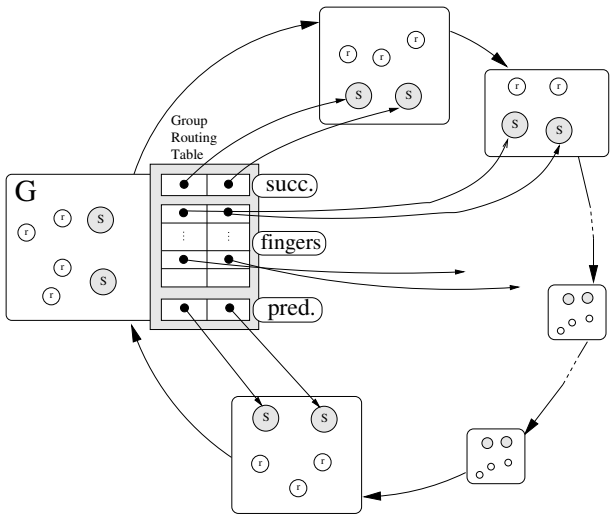


Figure 2.4: Group routing table in the top-level Chord overlay network.

Fingers improve the look-up performance, but are not necessary for successfully routing requests (a request can always be routed by following the successor pointers until it reaches its destination. However, this routing leads to $O(N)$ hops to reach the destination). Therefore, when the superpeers S_i of a group G_i change, we do not immediately update the fingers that point to G_i. Instead, we lazily update the finger tables when we detect that they contain invalid references (similarly to the lazy update of the fingers in regular Chord rings [33]). It is worth noticing that the regular Chord must perform a look-up operation to find a lost finger. Due to the redundancy that our multiple superpeer approach provides, we can choose without delay another superpeer in the finger vector for the same group, as long as there are still reachable superpeers left. A redundant Chord routing table may be implemented in a regular Chord DHT, by adding to the peer state the successors of each finger, for example. However, in that case we still have many more unstable than stable peers among each peer's neighbors, and the look-up speed improvement is balanced by an increased maintenance effort for larger routing tables. In our case, only peers with sufficient power need to maintain large routing tables.

To route a request to a group pointed to by a vector (successor or finger), we choose a random IP address from the vector and forward the request to that superpeer. When groups have more than one superpeer, this load balancing strategy avoids overloading specific nodes or communication links, and increases the scalability of the system.

2.4.3 Look-Up Latency With Hierarchical Chord

In this section, we quantify the improvement of the look-up latency due to the hierarchical organization of the peers. To this end, we compare the look-up performance of two DHTs. The first DHT is the flat Chord DHT. The second DHT is a two-tier hierarchy in which Chord is used for the top level overlay, and arbitrary DHTs are used for the bottom level overlays. For

each bottom level group, we only suppose that the peers in the group are topologically close so that intra-group look-up delays are negligible. We shall show that the hierarchical DHT can significantly reduce the look-up latency, owing to the heterogeneous availabilities of the peers and the reduction of peers in the Chord ring.

In order to make a fair comparison, we suppose that both the flat and hierarchical DHTs have the same number of peers, denoted by N. Let I be the number of groups in the hierarchical design. Because peers are joining and leaving the ring, the finger entries in the peers will not all be accurate. This is more than probable, since fingers are updated lazily. To capture the heterogeneity of the peers, we suppose that there are two categories of peers:

- *Stable peers*, for which each peer fails with probability p_s.
- *Instable peers*, for which each peer is unavailable with probability p_r, with $p_r \gg p_s$.

We suppose that the vast majority of the peers are instable peers. In real P2P networks, like Gnutella, most peers just remain connected the time of getting data from other peers. This fact has already been observed in [29]. Indeed, recent studies like [74] show a skewed heavy-tailed distribution (Pareto) for the time peers remain connected to the network in real P2P systems. For the hierarchical organization, we select superpeers from the set of stable peers, and we suppose there is at least one stable peer in each group. Because there are many more instable peers than stable peers, the probability that a randomly chosen Chord peer is down in the flat DHT is approximately p_r. In the hierarchical system, because all the superpeers are stable peers, the probability that a Chord peer is down is p_s.

To compare the look-up delay for flat and hierarchical DHTs, we thus only need to consider a Chord ring with N peers, with each peer having the same probability p_f of being down (failure probability). The flat DHT corresponds to $(N, p_f) = (N, p_r)$ and the hierarchical DHT corresponds to $(N, p_f) = (I, p_s)$. We now proceed to analyze the look-up of the Chord ring (N, p_f). To simplify the analysis, we assume the N peers are equally spaced on the ring, i.e., the distance between two adjacent peers is $\frac{2^m}{N}$. Our model implies that when a peer attempts to contact a peer in its finger table, the peer in the finger table will be down with probability p_f, except if this is the successor peer, for which we suppose that the finger entry is always correct (i.e., the successor is up or the peer is able to find the new successor. This assures the correct routing of look-up queries).

Given an initial peer and a randomly generated key, let the random variable H denote the number of Chord hops needed to reach the target peer, that is, to reach the peer responsible for the key. Let T be the random variable that is the clockwise distance in number of peers from the initial peer to the target peer. We want to compute the expectation $E[H]$. Assuming a uniform distribution of keys and a uniform key request probability[1], clearly

[1]This artifact serves the purpose of our analysis, but keys may be actually requested with very different probabilities, as will be seen in Chapter 6. We can assume a very efficient load-balancing technique implemented on the DHT to make our model more realistic.

$$E[H] = \sum_{n=0}^{N-1} P(T=n)E[H|T=n] = \frac{1}{N}\sum_{n=0}^{N-1} E[H|T=n] \tag{2.1}$$

From (2.1), it suffices to calculate $E[H|T=n]$ to compute $E[H]$. Let $h(n) = E[H|T=n]$. Note that $h(0) = 0$ and $h(1) = 1$. Let

$$j_n = \max\{j \,:\, 2^j \leq \frac{2^m n}{N}\} \tag{2.2}$$

The value j_n represents the number of finger entries that precede the target peer, excluding finger 0, the successor. For each of the finger entries, the probability that the corresponding peer is up is $1 - p_f$ (the successor is assumed always up, so Chord routing is assured to eventually succeed).

Starting at the initial peer, when hopping to the next peer, the query will advance $\lceil \frac{2^{j_n}}{2^m/N} \rceil$ peers if the j_nth finger peer is up; if this peer is down but the $(j_n - 1)$th finger peer is up, the query will advance $\lceil \frac{2^{j_n-1}}{2^m/N} \rceil$; and so on. Let $q_n(i)$ denote the probability that the ith finger is used. We therefore have

$$h(n) = 1 + \sum_{i=0}^{j_n} q_n(i) h\Big(n - \Big\lceil \frac{2^i}{2^m/N} \Big\rceil\Big) \tag{2.3}$$

The probability that the ith finger is used is given by

$$q_n(i) = p_f^{j_n - i}(1 - p_f) \;\; i = 1, \ldots, j_n \tag{2.4}$$

and by $q_n(0) = p_f^{j_n}$. Combining the above two equations 2.3 and 2.4 we obtain

$$h(n) = 1 + p_f^{j_n} h(n-1) + (1-p_f)\sum_{i=1}^{j_n} p_f^{j_n - i} h\Big(n - \Big\lceil \frac{2^i}{2^m/N} \Big\rceil\Big) \tag{2.5}$$

Using this recursion, we can calculate all the $h(n)$'s beginning at $h(0) = 0$. We then use (2.1) to obtain the expected number of hops, $E[H]$.

In Figure 2.5, we plot the expected number of hops in a look-up as a function of the availability of the peers in a Chord system, for different values of N. We can observe an exponential increase in the number of steps only for $p_f > 0.6$. With smaller values of p_f, although peers in the ring are not totally reliable, we are still able to advance quite quickly on the ring. Indeed, while the best finger for a target peer is unavailable with probability p_f, the probability of the second best choice to be also down is p_f^2, which is far smaller than p_f.

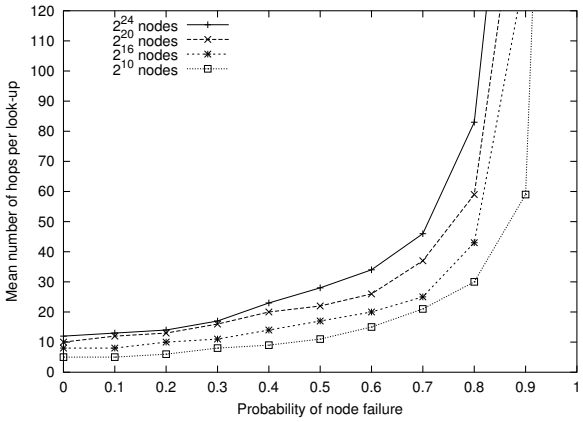


Figure 2.5: Mean number of hops per look-up in Chord.

	Flat		**Hierarchical**
	$p_r = 0.5$	$p_r = 0.8$	$p_s = 0$
$N = 2^{16}$ $I = 2^{10}$	17	43	5
$N = 2^{20}$ $I = 2^{16}$	22	59	8
$N = 2^{24}$ $I = 2^{20}$	28	83	10
$N = 2^{24}$ $I = 2^{16}$	28	83	8

Table 2.1: Comparing the flat and hierarchical networks.

Despite the good scalability of the Chord look-up algorithm in a flat configuration, the hierarchical architecture can yet significantly decrease the look-up delay. Table 2.1 gives the expected number of hops for the flat and hierarchical schemes, for different values of N, I, and p_r ($p_s = 0$). We suppose in all cases groups of $\frac{N}{I}$ peers. As an example, for $N = 2^{20} = 1,048,676$ peers, and $p_r = 0.8$, we observe that about 59 hops are needed on average. However, if we organize these peers in $I = 2^{16} = 65536$ groups of 16 peers each, the number of hops is reduced to 8. Since the number of steps is directly related to the look-up delay, we can conclude that the average look-up delay is divided by a factor of 7 in this case.

2.4.4 Hierarchical P2P System Implementation

We have developed a distributed file storage application based on the P2P paradigm, where the storage peers are organize using a DHT. The file storage application uses classic techniques like partitioning of a file in fixed-length data blocks and block replication, which do not deserve further comment in this work. Our first prototype featured plain standard Chord as the P2P look-up layer. Introducing the ideas of our ongoing research, we developed a second prototype using a hierarchical P2P look-up. We were able to keep the storage application almost unmodified, thanks to a careful layered protocol design.

The hierarchical look-up layer is a two-tier P2P system that uses Chord in the top level and CARP inside groups. As a consequence of this two-tier organization, the system is conceived to spread over several groups of reduced size. This is actually the scenario found in the VTHD [78] project, where several universities and research centers in France develop new applications across a Gigabit national IP network. We have deployed our prototype on hosts of the participating sites. Our prototype serves as a proof of concept demonstrating the advantages of a hierarchical organization and allows us to compare the performance of the flat and hierarchical approaches as part of our current and future work.

2.5 Conclusion

Hierarchical organizations in general improve overall system scalability. An example of a hierarchical organization is the Internet routing architecture with intra-domain routing protocols such as RIP or OSPF, and with BGP as inter-domain routing protocol. In this chapter we have proposed a generic framework for the hierarchical organization of peer-to-peer overlay network, and we have demonstrated the various advantages it offers over a flat organization. A hierarchical design offers higher stability by using more "reliable" peers (superpeers) at the top levels. It can use various inter- and intra-group look-up algorithms simultaneously, and treats join/leave events and key migration as local events that affect only a single group. By organizing peers into groups based on topological proximity, a hierarchical organization also generates fewer messages in the wide area and can significantly improve the look-up performance. Finally, as shown in Section 2.3, our architecture is ideally suited for caching popular content in local groups. By first querying the responsible peer within one's own group, popular objects are dynamically pulled into the various groups. This local-group caching can dramatically reduce download delays in peer-to-peer file-sharing systems. We revisit the caching feature of hierarchical P2P systems later in Section 3.3.

We have presented an instantiation of our hierarchical peer organization using Chord at the top level. The Chord look-up algorithm required only minor adaptations to deal with groups instead of individual peers. We have analyzed and quantified the improvement in look-up performance of hierarchical Chord. When all peers are available, a hierarchical organization reduces the length of the look-up path by a factor of $\frac{\log N}{\log I}$, where I is the number of groups and N is the total number of peers. When each peer is unavailable with a certain probability, a hierarchical organization reduces the length of the look-up path dramatically for the case where the failure probability of superpeers is significantly smaller than the failure probability of regular peers.

We have implemented a prototype of a two-tier P2P system, with Chord in the top level and CARP inside groups. Furthermore, a distributed file storage application is already functional and working on top of the look-up layer [79, 80].

Chapter 3

TOPLUS: Topology-Centric Look-Up Service

Slight not what's near through aiming at what's far.

Euripides. Rhesus

Topological considerations are of paramount importance in the design of a P2P look-up service. We present TOPLUS, a structured Peer-to-peer network based on the hierarchical grouping of peers according to network IP prefixes. TOPLUS is fully distributed and symmetric, in the sense of hierarchical DHTs where all peers have the same role. TOPLUS features a look-up service where queries are routed to their destination along a path that mimics the router-level shortest-path, thereby providing a small "stretch". Experimental evaluation confirms that a look-up in TOPLUS takes time comparable to that of IP routing.

3.1 Introduction

Several important proposals have recently been put forth for providing a distributed Peer-to-peer (P2P) look-up service, which are all based on *Distributed Hash Tables (DHT)*. It turns out that for many measures — like speed of look-up and potential for caching — it is highly desirable that the look-up service takes the underlying IP-level topology into account. Researchers have recently proposed modifications to the original look-up services that take topology into special consideration [65, 71, 70], or created Topology-aware Overlays [81].

In this chapter we explore the following issues: (1) How can we design a P2P look-up service for which topological considerations take precedence? (2) What are the advantages and disadvantages of such a topology-centric design? and (3) How can the topology-centric design

be modified so that the advantages of the original design are preserved but the disadvantages are abated?

To respond to the first question, we propose a P2P network design with a new look-up service, *Topology-Centric Look-Up Service (TOPLUS)*. In TOPLUS, peers that are topologically close are organized into groups. Furthermore, groups that are topologically close are organized into supergroups, and close supergroups into hypergroups, etc. The groups within each level of the hierarchy can be heterogeneous in size and in fan-out. The groups can be derived directly from the network prefixes contained in BGP tables or from other sources. TOPLUS has many strengths, including:

- *Stretch:* Later we define "stretch" as the relation of the latency between two points through layer-3 routing (IP) and using the overlay network. The more similar these latencies, the smaller the stretch. Packets are routed thought the overlay to their destination along a path that mimics the router-level shortest-path distance, thereby providing a small stretch.
- *Caching:* On-demand P2P caching of data is straightforward to implement, and can dramatically reduce average file transfer delays.
- *Efficient forwarding:* As we shall see, peers can use highly-optimized IP longest-prefix matching techniques to efficiently forward messages.
- *Symmetric:* Design lets peers similar responsibilities.

TOPLUS is an "extremist's design" to a topology-centric look-up service. At the very least, it serves as a benchmark against which other look-up services can compare their stretch and caching performance.

This chapter is organized as follows. We present related work at the end of this section. In Section 3.3 we describe the TOPLUS design, and we elaborate on its limitations and possible solutions in Section 3.4. In Section 3.5 we describe how we obtained the nested group structures from BGP tables and our measurement procedure for evaluating the average stretch. We then provide and discuss our experimental results. We conclude in Section 3.7.

3.2 Related Work

In [65], the authors show how the original CAN design can be modified to account for topological considerations. Their approach is to use online measurement techniques to group nodes into "bins". Yet the resulting stretch remains significant in their simulation results. The work in [82] on Internet network distance using coordinate spaces points to an interesting direction to explore in the context of P2P systems.

In [71], the authors examine the topological properties of a modified version of Pastry. In this design, a message typically takes small topological steps initially and big steps at the end of the route. We shall see that TOPLUS does the opposite, initially taking a large step, then a series of very small steps. Although [71] reports significantly lower stretches than other look-up services, it still reports an average stretch of 2.2 when the Mercator [83] topology model is used. Coral [84] has been recently proposed to adapt Chord to the Internet topology. Coral organizes peers in clusters and uses a hierarchical look-up of keys that tries to follow a path inside one peer's cluster whenever possible. The query is passed to higher-level clusters when the look-up can't continue inside the original cluster.

Cluster-based Architecture for P2P (CAP) [66] is a P2P architecture that also introduces topological considerations in its design. CAP is an unstructured P2P architecture whereas TOPLUS is a structured DHT-based architecture. Unlike CAP, TOPLUS uses a multi-level hierarchy and a symmetric design. Nevertheless, although TOPLUS does not mandate a specific clustering technique to create groups, we believe those of Krishnamurthy and Wang [85, 86] used in CAP are currently among the most promising.

This paper [87] presents a two-tier P2P network, as those seen in Chapter 2, that attempts to minimize the distortion between the look-up path of queries though the P2P network and their path over the network topology.

A DHT that adapts to the geographic location of peers in the network can be found in [88]. This work, however, is more oriented towards peers that are able to move freely, forming an ad-hoc network.

3.3 Overview of TOPLUS

Given a message containing key k, the P2P look-up service routes the message to the current up peer that is responsible for k. The message travels from source peer p_s, through a series of intermediate peers $p_1, p_2, \ldots, p_v$, and finally to the destination peer, p_d.

The principal goals of TOPLUS are as follows: (1) Given a message with key k, source peer p_s sends the message (through IP-level routers) to a first-hop peer p_1 that is "topologically close" to p_d; (2) After arriving at p_1, the message remains topologically close to p_d as it is routed closer and closer to p_d through the subsequent intermediate peers. Clearly, if the look-up service satisfies these two goals, the stretch should be very close to 1. We now formally describe TOPLUS in the context of IPv4.

Let I be the set of all 32-bit IP addresses[1]. Let $\mathcal{G}$ be a collection of sets such that $G \subseteq I$ for each $G \in \mathcal{G}$. Thus, each set $G \in \mathcal{G}$ is a set of IP addresses. We refer to each such set G as a

[1]For simplicity, we assume that all IP addresses are permitted. Of course, some blocks of IP addressed are private and other blocks have not been defined. TOPLUS can be refined accordingly.

group. Any group $G \in \mathcal{G}$ that does not contain another group in $\mathcal{G}$ is said to be an *inner group*. We say that the collection $\mathcal{G}$ is a *proper nesting* if it satisfies all the following properties:

1. $I \in \mathcal{G}$.
2. For any pair of groups in $\mathcal{G}$, the two groups are either disjoint, or one group is a proper subset of the other.
3. For each $G \in \mathcal{G}$, if G is not an inner group, then G is the union of a finite number of sets in $\mathcal{G}$.
4. Each $G \in \mathcal{G}$ consists of a set of contiguous IP addresses that can be represented by an IP prefix of the form $w.x.y.z/n$ (for example, $123.13.78.0/23$).

As shown later in Section 3.5, the collection of sets $\mathcal{G}$ can be created by collecting the IP prefix networks from BGP tables and/or other sources [85, 86]. BGP [45] (Border Gateway Protocol) is an inter-Autonomous System routing protocol. An Autonomous System (AS) is a routing infrastructure (networked routers) working under the same technical administration. Thus, an AS uses BGP to know which ASes it is connected to, and more important, which are the different ASes that can be reached through each of the neighboring ASes. Using BGP, an AS announces to other ASes the set of IP networks that can be reached through it. Each network is represented by an IP prefix, and prefix aggregation allows to hide the complexity of routing inside the AS, while offering the other ASes enough information to route IP packets.

In our case, many of the sets $\mathcal{G}$ would correspond to ASes, other sets would be subnets in ASes, and yet other sets would be aggregations of ASes. This approach of defining $\mathcal{G}$ from BGP tables require that a proper nesting is created. In order to reduce the size of the nodal routing tables, groups may be aggregated and artificial tiers may be introduced. Note that the groups differ in size, and in number of subgroups (the fanout).

If $\mathcal{G}$ is a proper nesting, then the relation $G \subset G'$ defines a partial ordering over the sets in $\mathcal{G}$, generating a partial-order tree with multiple tiers. The set I is at tier-0, the highest tier. A group G belongs to tier-1 if there does not exist a G' (other than I) such that $G \subset G'$. We define the remaining tiers recursively in the same manner (see Figure 3.1).

3.3.1 Peer State

Let L denote the number of tiers in the TOPLUS tree, let U be the set of all current up peers and consider a peer $p \in U$. Peer p is contained in a collection of telescoping sets in $\mathcal{G}$; denote these sets by $H_i(p), H_{i-1}(p), \cdots, H_0(p) = I$, where $H_i(p) \subset H_{i-1}(p) \subset \cdots \subset H_0(p)$ and $i \leq L$ is the tier depth of p's inner group. Except for $H_0(p)$, each of these telescoping sets has one or more siblings in the partial-order tree (see Figure 3.1). Let $\mathcal{S}_i(p)$ be the set of siblings groups of $H_i(p)$ at tier i. Finally, let $\mathcal{S}(p)$ be the union of the sibling sets $\mathcal{S}_1(p), \cdots, \mathcal{S}_i(p)$.

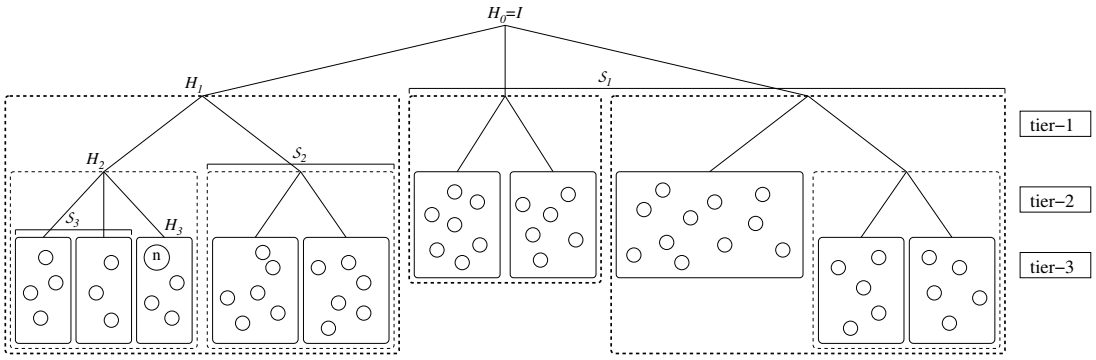


Figure 3.1: A sample TOPLUS hierarchy (inner groups are represented by plain boxes)

For each group $G \in \mathcal{S}(p)$, peer p should know the IP address of at least one peer in G and of all the other peers in p's inner group. We refer to the collection of these two sets of IP addresses as peer p's *routing table*, which constitutes peer p's state. The total number of IP addresses in the peer's routing table in tier i is $|H_i(p)| + |\mathcal{S}(p)|$.

3.3.2 XOR Metric

Each key k' is required to be an element of I', where I' is the set of all b-bit binary strings ($b \geq 32$ is fixed). A key can be drawn uniformly randomly from I', or it can be biased as we will describe later. For a given key $k' \in I'$, denote k for the 32-bit suffix of k' (thus $k \in I$ and $k = k_{31}k_{30}\ldots k_1k_0$). Throughout the discussion below, we will refer to k rather than to the original k'.

The XOR metric defines the distance between two IDs j and k as $d(j,k) = \sum_{\nu=0}^{31} |j_\nu - k_\nu| \cdot 2^\nu$. The metric $d(j,k)$ has the following properties:

- If $d(i,k) = d(j,k)$ for any k, then $i = j$.
- $\max d(j,k) \leq 2^{32} - 1$.
- Let $c(j,k)$ be the number of bits in the common prefix of j and k. If $c(j,k) = m$, $d(j,k) \leq 2^{32-m} - 1$.
- If $d(i,k) \leq d(j,k)$, then $c(i,k) \geq c(j,k)$.

$d(j,k)$ is a refinement of longest-prefix matching. If j is the unique longest-prefix match with k, then j is the closest to k in terms of the metric. Further, if two peers share the longest matching prefix, the metric will break the tie. The Kademlia DHT [89] also uses the XOR metric. The peer p' that minimizes $d(k,p)$, $p \in U$ is "responsible" for key k.

3.3.3 The Look-Up Algorithm

Suppose peer p_s wants to look up key k. Peer p_s determines the peer in its routing table that is closest to k in terms of the XOR metric, say p_j. Then p_s forwards the message to p_j. The process continues, until the message with key k reaches a peer p_d such that the closest peer to k in p_d's routing table is p_d itself. p_d is trivially the peer responsible for k.

If the set of groups form a proper nesting, then it is straightforward to show that the number of hops in a look-up is at most $L+1$, where L is the depth of the partial-order tree. In the first hop the message will be sent to a peer p_1 that is in the same group, say G, as p_d. The message remains in G until it arrives at p_d.

Each peer in TOPLUS mimics a router in the sense that it routes messages based on a generalization of longest-prefix matching of IP addresses, using highly-optimized algorithms [90].

3.3.4 Overlay Maintenance

When a new peer p joins the system, p asks an arbitrary existing peer to determine (using TOPLUS) the closest peer to p (using p's IP address as the key), denoted by p'. p initializes its routing table with p''s routing table. Peer p's routing table should then be modified to satisfy a "diversity" property: for each peer p_i in the routing table, p asks p_i for a random peer in p_i's group. This way, for every two peers in a group G, their respective sets of delegates for another group G' will be disjoint (with high probability). This assures that, in case one delegate fails, it is possible to use another peer's delegate. Finally, all peers in p's inner group must update their inner group tables.

Groups, which are virtual, do not fail; only peers can fail. Existing groups can be partitioned or aggregated on a slow time scale, as need be. When needed, keys can be moved from one group to another lazily: when a peer receives a query for a key that it is not storing, the peer can perform itself a query excluding its own group. Once the key is retrieved, queries can be normally satisfied.

3.3.5 Hierarchical Design

TOPLUS is one example of a hierarchical P2P network, as described in Chapter 2. A L-tier TOPLUS tree corresponds to a L-tier hierarchical DHT, with a symmetric design (see Section 2.3), where all peers are superpeers. At each tier (from the first to the bottom-most) a CARP [72]-like DHT using the XOR metric is employed to resolve the responsible group for a key. In the last step of the look-up procedure, the same algorithm is used to find the peer responsible for the key in the destination inner group.

Superpeers are not used *explicitly*, but there is an implicit research of the most stable peers through the diversity property conservation procedure: if a peer p connects to a unstable delegate in group G, the delegate is bound to leave the system soon, and thus p must ask for another delegate in that group G. Peer p changes the delegate until a stable one is chosen. Peers who had from the beginning connected to stable delegates have not a reason to change them. Thus, in the end, the most stable peers play a tacit role of superpeers, but avoiding a explicit selection algorithm as presented in Chapter 2. Again, we see that P2P networks tend to build around the most stable peers [25].

3.3.6 TOPLUS Network Initialization

We have just described how a new peer can join an existing TOPLUS network, but how is this network created in the first place? We show a method to initialize the P2P network, which validates the TOPLUS algorithms even in the early stages of deployment. Our solution involves the existence of one first peer that knows the whole TOPLUS tree. This is not a special requirement: recall that the TOPLUS tree is calculated off-line, and its knowledge is necessary *before* the TOPLUS network is deployed. This peer is called the "primordial peer", or p^0, for short.

p^0 can be deployed by some institution, company or individual offering a service involving TOPLUS. p^0 should be a powerful and well-connected host. If needed, p^0 can be implemented by a cluster of machines, provided they act as a coordinated entity.

When the system is first deployed, no peer can be found in any group. However, p^0 is the first "virtual" member of each tier-1 group. Thus, p^0 has a routing table where the delegate in all tier-1 groups is itself.

When a first peer p comes into the system, it must contact the only existing peer, p^0. p asks p^0 to find p's corresponding inner group G, and the contact peer in G is p^0 again. However, when p^0 in G provides the new peer p with the routing table, p^0 is not among the members of group G: p is alone in G. Then p, to maintain the "diversity property", asks each peer in the routing table for an alternate delegate. If no new peer in the whole network has joined, p will remain with p^0 as the only delegate in each group. Instead, if one peer q came to any other group G', p^0 must know about him, and q is returned as an alternate delegate in G'.

Note that arriving peers cannot in the future use p^0 when they answer to a query for a peer in their group to satisfy the diversity property: when a peer receives the list of members of its group, the primordial peer is never among them. Thus, p^0 silently disappears from the system over time.

Of course, we are assuming here that once the system starts its deployment, there is a continuous flow of arriving peers. If the first peer arrives and afterwards leaves, p^0 should come back to the initial situation. We also assume that the peer population spreads more or less uniformly over all groups. That should be the case at least in statically deployed TOPLUS trees. In other

cases, the TOPLUS tree should contain only groups where peers indeed exist or may exist (e.g., personal dial-up and ADSL ISPs for file-sharing applications), eventually reaching a situation where all groups are more or less populated.

Recall that each peer must know another in each tier-1 group. The primordial peer avoids communication to all tier-1 groups each time one of them is populated for the first time. There's always somebody in all tier-1 groups: p^0, at the very least. Peers just get to know new peers in other groups, through the "diversity property" conservation process, which is less costly than broadcasting to all groups the arrival of a peer to a previously empty group.

This approach can be introduced in other DHTs. Although the technical details may differ because of the different algorithms, the idea is to initialize the P2P network with a set of virtual peers uniformly distributed in the ID space. Those peers are implemented by a single physical host (or cluster), namely our p^0. Our technique gives stability to the initial network.

3.3.7 On-Demand P2P Caching

TOPLUS can provide a powerful caching service for an ISP. Suppose that a peer p_s wants to obtain the file f associated with key k, located at some peer p_d. It would be preferable if p_s could obtain a cached copy of file f from a topologically close peer.

To this end, suppose that some group $G \in \mathcal{G}$, with network prefix w.x.y.z/r, at any tier, wants to provide a caching service to the peers in G. Further suppose all pairs of peers in G can send files to each other relatively quickly (high-speed LAN) or at least quicker than to any other peer out of G. Now suppose some peer $n_s \in G$ wants to find the file f associated with key $k \in I$. Then p_s creates a new key, k_G, which is equal to k but with the first r bits of k replaced with the first r bits of w.x.y.z/r. Peer p_s then inserts a message with key k_G into TOPLUS. The look-up service will return to p_s the peer n_G that is responsible for k_G. Peer p_G will be in G, and all the messages traveling from p_s to p_G will be confined to G. If p_G has f (cache hit), then p_G will send f to p_s at a relatively high rate. If p_G does not have f (cache miss), p_G will use TOPLUS to obtain f from the global look-up service. After obtaining f, p_G will cache f in its local shared storage and pass a copy of f to p_s. The techniques in [91] can be used to optimally replicate files throughout G to handle intermittent nodal connectivity. TOPLUS can implement a hierarchical Internet cache of the kind described in [92].

3.4 Drawbacks and Solutions

The price to pay for TOPLUS' features is the sacrifice of other desirable properties in a P2P look-up service. We now discuss some of the drawbacks of the TOPLUS design, and approaches to address them.

Non-uniform population of ID space: The number of keys assigned to an inner group will be approximately proportional to the number of IP addresses covered by the inner group, not to the number of peers available in the group. Some peers may be responsible for a disproportionate number of keys: when a group is defined by a short IP prefix, many keys are mapped to the peers inside if keys are generated uniformly over the IP address space. Thus, if few peers happen to be inside said group, they will be overloaded with keys.

Lack of virtual peers: CAN, Chord, Pastry and Tapestry can assign virtual peers to the more powerful peers, so that they get more keys to store in the P2P storage system implemented on top. TOPLUS, as currently defined, does not facilitate the creation of virtual peers.

Correlated peer failures: As with Chord, Pastry and Tapestry, TOPLUS can replicate key/data pairs on successor peers within the same inner group. However, if an entire inner group fails (e.g. link failure), then all copies of the data for the key become unavailable.

One first enhancement to TOPLUS is to use a non-uniform distribution when creating keys. Specifically, suppose a peer knows about J inner groups, and we estimate the average fraction of active peers in inner group j to be q_j. Then when assigning a key, we first choose an integer (deterministically) from $\{1,2,\ldots,J\}$ biasing the decision with the weights $w_1,\ldots,w_J$. The larger the q_j of a group j, the higher the corresponding weight w_j. Suppose integer j is selected, and group j has prefix w.x.y.z/n. We then choose a key uniformly from the set of addresses covered by w.x.y.z/n. Still, this method requires a peer population which remains stable, although individual peers may change over time.

To address the lack of virtual peers, we assign each peer a permanent "virtual ID" uniformly distributed over the address space of the peer's inner group. More powerful peers are assigned multiple permanent virtual IDs, thereby creating virtual peers. In the inner group table, we list the virtual IDs associated with each IP address. Virtual IDs are generated by assigning different $(m-32)$-bit string to each peer, with the remaining 32 bits fixed by the IP address of the peer. We modify TOPLUS as follows: first the message reaches a peer p such that the longest prefix match is inside the inner group of p. Then p determines in the inner group the virtual ID that is the closest to the key. The responsible peer is the virtual ID's owner.

To solve the problem of correlated peer failures in a P2P storage system, when we replicate key/data pairs, we need to distribute the replicas over multiple inner groups. Inner group failures must be detected. We only sketch a partial solution here: we use K distinct hash functions to create keys, generating K different locations that are geographically dispersed with high probability.

3.5 Benchmarking TOPLUS

In TOPLUS, a group is defined by an IP network prefix. We have used IP prefixes obtained from two sources: the BGP routing tables of routers in the Internet, and the prefixes of well-known networks (such as corporate LANs or ISPs). As shown in [86, 66], the IP prefixes obtained from BGP routing tables form clusters of hosts that are topologically close to each other. Our assumption is that this locality property is recursively preserved to some extent in coarser IP prefixes that regroup clusters (super-clusters, clusters of super-clusters, etc)..

IP network prefixes were obtained from several sources: BGP tables provided by Oregon University [93] (123,593 prefixes) and by the University of Michigan and Merit Network [94] (104,552); network IP prefixes from routing registries provided by Castify Networks [95] (143,082) and RIPE [96] (124,876). After merging all this information and eliminating reserved and non-routable prefixes, we have obtained a set of *250,562* distinct IP prefixes that we organize in a partial order tree (denoted as Prefix Tree hereafter). Studies from NLANR [97] show that about 20% of the total usable IP space was routable in 1997, and 25% in 1999. As our tree covers 35% of the IP space, we consider it a valid snapshot of the Internet in 2003.

Source	**IP prefixes provided**
Oregon University	123,593
Michigan U. and Merit Network	104,552
Castify Networks	143,082
RIPE Routing Registry	124,876
Total distinct IP prefixes	*250,562*

Table 3.1: IP prefixes obtained from different sources

3.5.1 Measuring Stretch

The stretch is defined as the ratio between the average latency of TOPLUS routing (using the Prefix Tree) and the average latency of IP routing. Ideally, we could use the `traceroute` [98] tool to measure the delay between arbitrary hosts in the Internet.[2] However, security measures deployed in almost every Internet router nowadays prevent us from using this simple and accurate measurement technique. Therefore, we have used the `king` [99] tool to obtain experimental results. `king` gives a good approximation of the distance between two arbitrary hosts by measuring the latency between the DNS servers responsible for these hosts. For a more in-depth study on the precision of `king` measurements, we refer the reader to Chapter 4.

The general principle of our measurements is shown in Figure 3.2, where peer at address

[2]This can be achieved specifying one of the two host as a gateway packets must pass through.

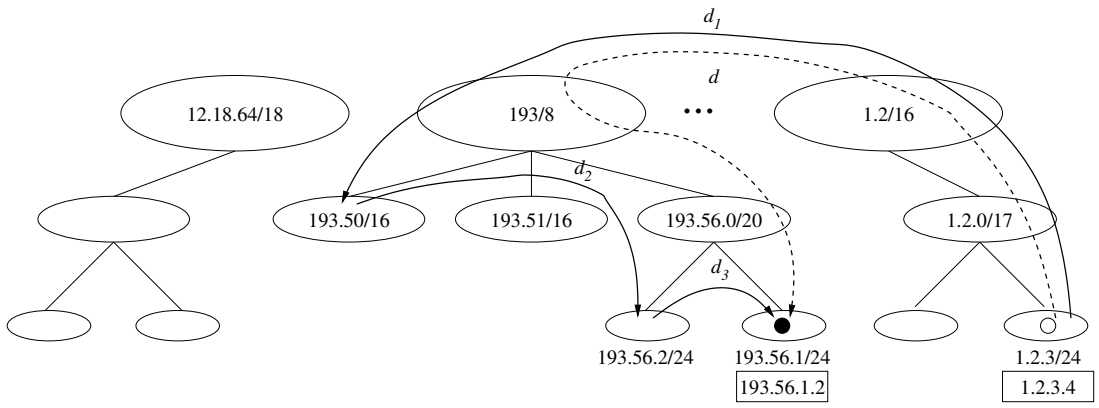


Figure 3.2: Path followed by a query in the Prefix Tree

1.2.3.4 sends a query for key k, whose responsible peer is 193.56.1.2. Following the TOPLUS routing procedure, peer 1.2.3.4 must first route the query inside the tier-1 group containing k. Peer 1.2.3.4 selects a delegate from its routing table in group $193/8$, because $k/32 \subset 193/8$ (note that there cannot be another group G in tier-1 satisfying $k/32 \subset G$). Assuming the delegate is in group $193.50/16$, the query is first routed along the path labeled with latency d_1 in Figure 3.2. Then, the delegate selects the (unique) tier-2 group inside $193/8$ which contains k: $193.56.0/20$. Let the new delegate peer be in tier-3 group $193.56.2/24$. The query is forwarded to that delegate peer along path d_2. Finally, the destination group $193.56.1/24$ is reached with the next hop d_3 (as we did in our analysis of Chapter 2, we neglect the final forwarding of the query inside the destination group). In contrast to TOPLUS routing, the query would follow path d between 1.2.3.4 and 193.56.1.2 with IP routing. The stretch for this particular case would be $\frac{d_1+d_2+d_3}{d}$.

In general, we consider the length of the path from one peer to another as the weighted average of the length of all possible paths between them. Path weights are derived from the probability of a delegate peer to be in each of the different groups at each tier. Assuming a uniform distribution of peers in the Internet, the probability of peer p choosing a delegate in group G at tier-i under parent group $S \in \mathcal{S}_{i-1}(p)$ can be computed as the number of IP addresses in all inner groups descendant of G divided by the number of IP addresses in all inner groups descendant of S. To simplify computations and keep them local to tier-i, we approximate this probability by computing the space of IP addresses covered by the network prefix of G, divided by the space of IP addresses covered by all groups children of S (including G). For instance, for the first hop in Figure 3.2, the probability of the delegate peer being in group $193.50/16$ is $\frac{2^{16}}{2^{16}+2^{16}+(2^{24}+2^{24})} = 0.498$; using the simplified formula, the probability is approximated as $\frac{2^{16}}{2^{16}+2^{16}+2^{12}} = 0.48$.

Consider a query issued by peer p_s in inner group S for a key k owned by peer p_d in inner group D at tier-i. Let $d^T(G,G')$ be the TOPLUS latency between a peer in group G and a peer in group G' and $d^{IP}(G,G')$ be the corresponding direct IP latency. Let $H_i,\ldots,H_0$ (as in Section 3.3.1) be the telescoping set of groups containing peer p_d (hence $H_0 = I$ and $H_i = D$).

To reach p_d, p_s forwards its request to its delegate peer p_g belonging to one of the inner groups G in H_1. Hence:

$$E[d^T(S,D)] = \sum_{G \subset H_1} P(G)(E[d^{IP}(S,G)] + E[d^T(G,D)]) \tag{3.1}$$

where $P(G)$ is the probability of p_g being in G. Thus, $P(G) = \frac{|G|}{|H_1|}$. Note that $E[d^{IP}(S,G)] = 0$ if ever $p_s \in H_1$, since in this case p_s is its own delegate in H_1. The process continues with p_g forwarding the query to its delegate peer p'_g in one of the inner groups G' in H_2. The equation for $E[d^T(G,D)]$ is thus similar to Equation (3.1):

$$E[d^T(G,D)] = \sum_{G' \subset H_2} P(G')(E[d^{IP}(G,G')] + E[d^T(G',D)]) \tag{3.2}$$

where $P(G') = \frac{|G'|}{|H_2|}$. A total of i recursions allow to obtain the value of $E[d^T(S,D)]$.

To obtain the average stretch of the Prefix Tree, we compute the stretch from a fixed origin peer to 1,000 randomly generated destination IP addresses using `king` to measure the delay of each hop. We compute the path length from the origin peer to each destination peer as the average length of all possible paths to the destination, weighted according to individual path probabilities (as described above). Finally, we compute the stretch of the Prefix Tree as the average stretch from the 1,000 previous measurements. For the experiments, we chose an origin peer at Institut Eurécom with IP address 193.55.113.1. We used 95% confidence intervals for all measurements. We detail now the different Prefix Tree configurations that we have considered in our experiments.

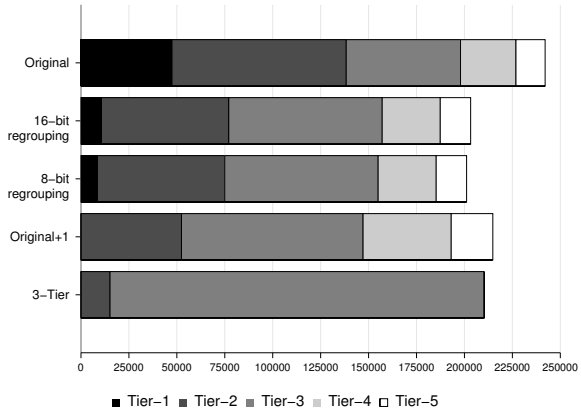
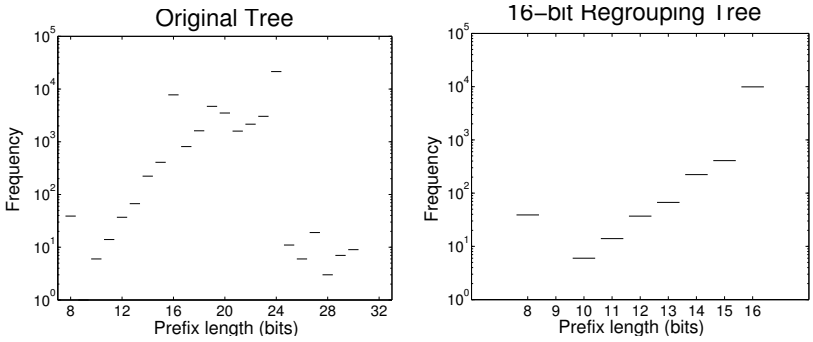


Figure 3.3: Number of groups per tier in the Prefix Trees

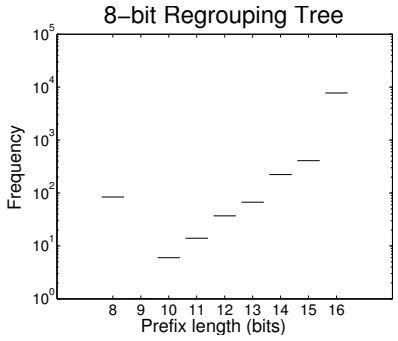
Original Prefix Tree

Original Prefix Tree is the tree resulting from the ordering of the IP prefixes using operator $\subset$. The partial order tree has 47,467 different tier-1 groups (prefixes); 10,356 (21.8%) of these groups have at least one tier-2 subgroup; further, 3,206 (30%) of tier-2 groups have at least one tier-3 subgroup. The deepest nesting in the tree comprises 11 tiers. The number of non-inner group at each tier decreases rapidly with the tier depth (following roughly a power-law [100]) and the resulting tree is strongly unbalanced.

Figure 3.4(a) shows the distribution of prefix lengths in tier-1. As most prefixes of more that 16 bits do not contain nested groups, the Original Prefix Tree has a large number of tier-1 groups. Consequently, the routing tables of each peer will be large because they have to keep track of one delegate per tier-1 group (See Figure 3.3). On the other hand, since 61% of the IP addresses covered by the tree are within tier-1 inner groups, a large number of peers can be reached with just one hop.



(a) Original. (b) 16-bit regrouping.



(c) 8 bit regrouping.

Figure 3.4: Prefix length distribution for the tier-1 groups of the three Prefix Trees.

We computed an average stretch of *1.17* for the Original Prefix Tree, that is, a query in TOPLUS takes on average 17% more time to reach its destination than using direct IP routing.

	stretch TOPLUS vs. IP (± confidence interval)		
Tier	Original	16-bit regroup.	8-bit regroup.
1	1.00 (±0.00)	1.00 (±0.00)	1.00 (±0.00)
2	1.29 (±0.15)	1.32 (±0.14)	1.56 (±0.23)
3	1.31 (±0.16)	1.30 (±0.17)	1.53 (±0.23)
4	1.57 (±0.50)	1.41 (±0.20)	1.56 (±0.50)
Mean	1.17 (±0.06)	1.19 (±0.08)	1.28 (±0.09)

Table 3.2: Stretch obtained in each tree, depending on the tier of the destination peer

In Table 3.2 we present the average stretch for IP addresses located in inner groups at tiers 1 to 4. As expected, we observe that the deeper we go in the tree to reach a destination, the higher the stretch becomes (because there is a higher probability of making more hops). More than half of the queries had for destination a peer in a tier-2 group.

Modified Prefix Trees

As previously mentioned, a large number of groups are found in tier-1 and all peers in the network must know a delegate in each of those groups. In order to reduce the size of the routing tables, we modify the tree by "aggregating" small groups that have a long prefix into larger groups not present in our IP prefix sources. We consider groups to be "small" if their prefix is longer than 16 bits; this represents 38,966 tier-1 groups for our experimental data. The graphic in Figure 3.3 compares the distribution of groups per tier for some of the trees we are about to study.

16-bit regrouping: A first approach consists in aggregating small groups into 16-bit aggregate prefix groups. This means that any tier-1 prefix `a.b.c.d/r` with $r > 16$ is moved to tier-2 and a new 16 bit prefix `a.b/16` is inserted at tier-1. This process creates 2,161 new tier-1 groups, with an average of 18 subgroups in each of them. The distribution of tier-1 prefixes is shown in Figure 3.4(b).

The resulting tree contains 10,709 tier-1 groups, 50% (5,454) of which contain subgroups. We have measured an average stretch of *1.19* for that tree (see Table 3.2). These results indicate that 16-bit regrouping essentially preserves the low stretch.

8-bit regrouping: We have experimented with a second approach to prefix regrouping, which consists in using coarser, 8-bit aggregate aggregate prefix groups. Any tier-1 prefix `a.b.c.d/r` with $r > 16$ is moved to tier-2 and a new 8 bit prefix `a/8` is inserted at tier-1 (if it does not already exist). This process creates 45 new tier-1 groups, with an average of 866 subgroups in each of them. The distribution of tier-1 prefixes is shown in Figure 3.4(c).

The resulting tree contains 8,593 tier-1 groups (more than 5 times less than our Original

Prefix Tree). 38% (3,338) of these groups contain subgroups and almost half of tier-2 groups (1,524) have again subgroups. The tree is clearly becoming more balanced and, as a direct consequence of the reduction of tier-1 groups, the size of the routing table in each peer becomes substantially smaller. We have measured an average stretch of *1.28* for the new tree (see Table 3.2). This remarkable result demonstrates that, even after aggressive aggregation of large sets of tier-1 groups into coarse 8-bit prefixes, the low stretch property of the original tree is preserved.

3.5.2 Routing Table Size

The principal motivation for prefix regrouping is to reduce the size of the routing tables. We estimate the size of the routing tables by choosing 5,000 random uniformly distributed IP addresses (peers); for each of these peers p, we examine the structure of the tree to determine the sibling sets $\mathcal{S}(p)$ and the inner group peers $H_N(p)$, and we compute the size $|\mathcal{S}(p)| + |H_N(p)|$ of the routing table of peer p.

Table 3.3 shows the mean routing table size depending on the tier of the peer, as well as the average. The route table size is mainly determined by the number of tier-1 groups. If we eliminate their delegates from the routing table, the size of the routing tables needed to route queries *inside* each tier-1 group remains small. Even using 8-bit regrouping, routing tables count more than 8,000 entries (see Table 3.3). To further reduce the routing table sizes, we transform the *Original* and the *16-bit-regrouping* trees such that all tier-1 prefixes are 8-bit long, which will limit the number of tier-1 groups to *at most* 256. We refer to the resulting trees as *Original+1* and *16-bit+1*. For this purpose, any tier-1 prefix `a.b.c.d/r` with $r > 8$ is moved to tier-2 and a new 8 bit prefix `a/8` is inserted at tier-1 (if it does not already exist).

	Mean routing table size						Mean routing table size		
Tier	Original		16-bit regroup.		8-bit regroup.		Original+1	16-bit+1	3-Tier
1	47,467	0	10,709	0	8,593	0	143	143	143
2	47,565	98	10,802	93	8,713	120	436	223	248
3	47,654	187	10,862	153	8,821	228	831	288	261
4	47,796	329	11,003	294	8,950	357	1,279	428	-
5	47,890	423	11,132	423	9,016	423	696	556	-

Table 3.3: Mean routing table size in each tree depending on the tier of a peer. For tree entries with two columns, the left column is the full routing table size and the right column is the size without tier-1 groups

We finally create another tree called *3-Tier* that has no more than 3 tiers. The top tier is formed by up to 256 groups with 8-bit long prefixes, tier-2 by up to 256 groups with 16-bit long prefixes, and the third tier by up to 256 groups each with a 24-bit long prefix. The mean routing table sizes for these three trees, presented in Table 3.3, on the right, shows dramatic reduction in the number of entries that must be stored by each peer. However the stretch is significantly

penalized, as shown in Table 3.5.2. We clearly face a tradeoff between look-up latency and memory requirements [32].

	Original+1	16-bit+1	3-Tier
stretch (TOPLUS/IP) / confidence margin	1.90 / (±0.20)	2.01 / (±0.22)	2.32 / (±0.09)

Table 3.4: TOPLUS vs. IP stretch in the trees where the tier-1 groups all have 8-bit prefixes.

3.6 Locality in a Hierarchical DHT

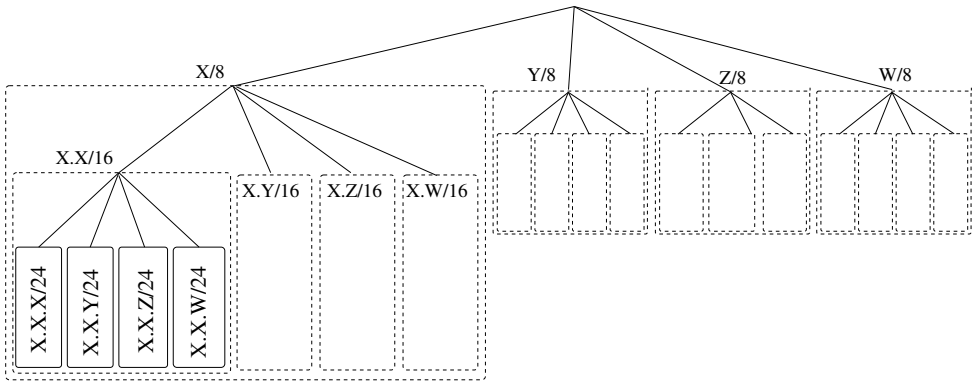
As we saw in Chapter 2, it is essential for an efficient hierarchical DHT to keep intra-group look-up delays low. It is easy to provide fast communication between peers within a group confined to a high-speed LAN. This situation corresponds to the ideal framework where our two-tier model best fits. However, some groups may span an entire ISP or one or several ASes. In those cases, we must try to build the DHT in a such a way that intra-group delay is *as low as possible*. Also, very large groups may contain a very large number of peers. A scalable approach is required to organize all the peers in such a group. We can solve both problems by organizing the peers in the general *n*-tier DHT hierarchy, such that each group at each tier comprises peers with good locality properties.

We have observed in Table 3.2 that the deeper the destination in the hierarchy, the higher the stretch. This is unsurprising, since a look-up through multiple tiers introduces additional hops from one group to another, increasing the possible divergence between the IP and overlay routing paths. However, those hops do not increase too much the stretch because they are made, at each tier, among groups that are close to each other.

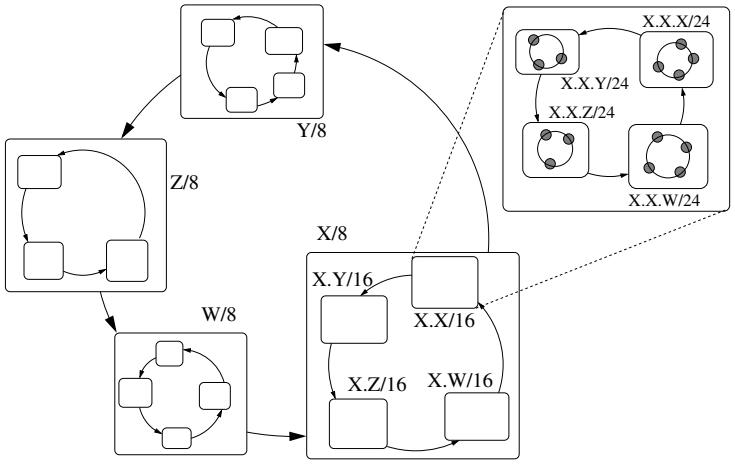
It is straightforward to build a proper nesting of groups from IP network prefixes inside a top-level group, and TOPLUS has demonstrated that this is a valid method to obtain efficient communication inside groups. Peer groups based on IP prefixes is an effective mechanism to construct hierarchical P2P networks, independently of the look-up algorithm used within groups. Figure 3.5 shows a mapping of TOPLUS groups to a 3-tier hierarchy of Chord rings. At each level, we obtain the best locality achievable inside and among groups. This validates our assumption of a hierarchy of groups of local peers.

3.7 Conclusion

TOPLUS takes an extreme approach for integrating topological consideration into a P2P network. TOPLUS is fully distributed, and is also symmetric in the sense that all peers have the



(a) A sample TOPLUS hierarchy with several groups and their associated IP prefixes.



(b) The mapping of those groups to a 3-tier hierarchical Chord (group with IP prefix $X.X/16$ detailed).

Figure 3.5: The mapping of a Hierarchical DHT on a TOPLUS hierarchy.

same role. TOPLUS bears some resemblance to Pastry [41, 71] and Tapestry [42]. In particular, Pastry and Tapestry also use delegate peers and prefix (or suffix) matching to route messages. However, unlike Pastry, we map the groups directly to the underlying topology, resulting in an unbalanced tree partitioning the IP address space in groups of very different sizes. The resulting routing scheme initially makes big physical jumps rather than small ones. We have shown that TOPLUS offers excellent stretch properties, resulting in a very fast look-up service. Although TOPLUS suffers from some limitations, which we have exposed and discussed, we believe that its remarkable speed of look-up and its simplicity make it a promising candidate for large-scale deployment in the Internet.

Chapter 4

MULTI+: Peer-to-Peer Topology-Aware Multicast Trees

Virtue is not left to stand alone. He who practices it will have neighbors.

Confucius

We have just presented TOPLUS, a structured peer-to-peer network that is based on the hierarchical grouping of peers according to network IP prefixes. In this chapter we introduce MULTI+, an application-level multicast protocol for content distribution over a P2P TOPLUS-based network. We use the characteristics of TOPLUS to design a protocol that allows for every peer to connect to an available peer that is close. MULTI+ trees also reduce the amount of redundant flows leaving and entering each network, making more efficient bandwidth usage. We have used different procedures to measure or compute the distances among peers, in order to validate the algorithms in MULTI+.

4.1 Introduction

IP multicast seems (or, at least, was designed) to be the ideal solution for content distribution over the Internet: (1) it can serve content to an unlimited number of destinations, and (2) uses the network bandwidth very efficiently. These two characteristics are strongly correlated. IP multicast saves bandwidth because a single data flow can feed many recipients. The data flow is only split at routers where destinations for the data are reached via more than one outgoing port. Thus n clients do not need n independent data flows, which allows for IP multicast's scalability. However, IP multicast was never widely deployed in the Internet: security reasons (e.g., any host can send to a given multicast group without being member of that group), the lack of congestion control [101, 102] (which makes sending data to heterogeneous receivers difficult),

problems with software implementation at the routers, charging conflict between peering ASes, and the fact that one packet entering a port can produce an undetermined number of outgoing packets, prevented IP multicast from being universally deployed. There is however a multicast service offered by Sprint or SIXXS, but in general IP multicast has not reached far outside academic environments, let alone integrated the everyday experience of the regular users of the Internet.

Lately, with the advent of broadband links like ADSL and the generalization of LANs at the workplace, the *edges* of the Internet started to see their bandwidth increasing. Together with the ever-cheaper and yet more powerful machines (computational power, storage capacity), millions of hosts connected to the Internet have the possibility of implementing themselves services at the application level that augment the capabilities of the network: the Peer-to-Peer (P2P) systems. Various application-level multicast implementations have been proposed [103, 104, 105, 106, 62], most of which are directly implemented on top of P2P infrastructures (Chord [33], CAN [40] or Pastry [41]). The good scalability of the underlying P2P networks gives these application-level multicast one of the properties of the original IP multicast service, that of serving content to a virtually unlimited number of clients (peers). However, these P2P networks are generally conceived as an application layer system completely isolated from the underlying IP network.

Thus, the P2P multicast systems that we know may fail at the second goal of IP multicast, namely efficiency: a LAN hosting a number of peers in a P2P multicast tree may find its outbound link saturated by *identical* data flowing to and from its local peers, unless those peers are somehow *aware* of the fact that they are sharing the same network. This situation remains valid for higher-tier networks (e.g., ISPs): even if the capacity of links to the Internet (as we have seen in Section 1.4.3) is designed using conservative assumptions, normally the resulting system does not allow for all or many hosts to maintain long-term traffic flows from and to the Internet. This is a matter of concern for ISPs due to P2P file-sharing applications and flat-rate commercial offers that make possible to be downloading content from a home computer 24 hours a day. This problem also affects application-level multicast sessions if peers do not take the network topology into consideration.

We have based our P2P multicast protocol on TOPLUS because of its inherent topology-awareness. We aim at building a P2P multicast network that avoids network link stress while remaining scalable. In TOPLUS, peers that are topologically close are organized into groups. In turn, groups that are topologically close are organized into supergroups, and nearby supergroups into hypergroups, etc. The groups within each level of the hierarchy can be heterogeneous in size and in fan-out (i.e., number of subgroups; the same terminology is used for peers as introduced in Section 1.3). TOPLUS derives the groups from the network prefixes contained in BGP tables or from other sources.

Before we elaborate on a multicast deployment in a TOPLUS network, we must state some assumptions about the system: First of all, there is a large population of peers participating in the network, which justifies the utilization of multicast in the first place. By large population we mean one that cannot get a satisfactory service using unicast over the given infrastructure

(e.g., 100 clients downloading content concurrently from a machine featuring a low-range 128 Kbps ADSL up-link). Second, the TOPLUS multicast infrastructure may be used to transmit to many different multicast groups, and thus many multicast trees must live together without interfering each other. Third, a peer is only willing to transmit data from a multicast tree if the peer is in the multicast group (that is, a peer will not use its bandwidth just to forward data for others). However, when participating in the TOPLUS multicast infrastructure, a peer must accept to cooperate with others in low-bandwidth maintenance tasks.

This chapter is organized as follows: Section 4.2 presents the related work in the field. Section 4.3 introduces MULTI+ and its main algorithms. Then we benchmark multicast trees created with MULTI+. We use two different methods to evaluate inter-host latency: `king` in Section 4.4 and a Euclidean n-dimensional coordinate space where each peer is represented as a point in Section 4.5. We present our conclusions in Section 4.6.

4.2 Related Work

Application-level multicast has given some interesting results like Narada [103] (based on the NICE project) or End System Multicast [104]. Narada organized peers in a fashion similar to ours, using hierarchical clustering of peers. Among those using overlay networks, we find examples using CAN [106] and Pastry (Scribe) [62]. Bayeux [105] is another overlay multicast protocol based on Tapestry [42]. An interesting comparative of these approaches can be found in [107]. Further examples of Application-level multicast are [108] or Yoid [109].

Content distribution overlay examples are SplitStream [110] and [111]. Recently, the problem of data dissemination on adaptive overlays has been treated in [112]. In [113], the authors present a solution to the mismatch between the overlays and the Internet infrastructure based on intensive distributed probing. Our approach differs mainly in the achievement of efficient topology-aware multicast trees with no or very little active measurement.

The authors in [114] build a topology-aware multicast overlay. The connections among peers are optimized by detecting overlap in routes to a sender. Their aim is to limit the number of identical packets on the same link, for better bandwidth utilization. They use `traceroute` and information in topology servers in OSPF to build optimal multicast tree. Still, they repeatedly probe the network for constant adaption to changing conditions, specially upon the arrival of new peers.

The approach in [115] attempts a geometric construction of minimal delay trees on a Euclidean space, where each peer is represented by a set of coordinates. Although the results are encouraging, their method depends on the obtention of a valid coordinate space, issue that remains open.

The authors of [116] and [117] propose adaptive overlays that attempt to fit one or more

constrains, like bandwidth or delay. The authors rely heavily on an anti-entropy (gossip-like) distribution of membership information called RanSub, allowing each peer to progressively know of an important fraction of the total peer population. Only when an admittedly partial, but extensive knowledge of other peers is achieved can the overlay perform a distributed probing that provides enough information to comply with the chosen metrics' constraints. We guess that the effect of massive peer failure (an aspect to be studied on Chapter 5) on this kind of network would be important, due to the time that membership changes would take to propagate through the anti-entropy mechanism.

In Bullet [118], peers form a mesh instead of a tree for content distribution. The main idea is increase the available bandwidth to each peer by downloading from multiple parents instead of just one, as is the case for trees. Basically, each peer benefits from parallel download [119] in a similar fashion as SplitStream does. However, Bullet is also based on RanSub. Moreover, the fact that each peer is able to download different chunks of data from different peers makes Bullet more suitable for large file distribution (like BitTorrent [120]) than for live content streams.

ZIGZAG [121] offers low root-to-leaf delay by building trees with few levels. This approach organizes peers in clusters, and the arrival of new peers may make a cluster split, with the subsequent overhead to reorganized the P2P network. ZIGZAG is not topology-aware, and the stretch (see Chapter 3) between direct IP routing paths from root to leafs and those of the application-level multicast is in average around 3.5. ZIGZAG can incur in relatively high link stress (many identical copies flow through the same physical link).

4.3 MULTI+: Multicast over TOPLUS

4.3.1 A Multicast Tree

In a first approach we assume that all peers, including the source of the multicast tree, are connected through links where the available bandwidth is not a constraint, i.e., there is excess of bandwidth available for the application. A simple multicast tree is shown in Figure 4.1. Let S be the source of the multicast group m. Of course, the source should be located in some group of a TOPLUS tree. This latter would be covering the whole Internet. Peer p is receiving the flow from peer q. We say that q is the parent of p in the multicast tree. Conversely, we say that p is a child of q, its parent in the multicast tree. Peer p is in level-3 of the multicast tree and q in level-2. It is important to note that, in principle, the *level* where a peer is in the multicast tree has nothing to do with the *tier* the peer belongs to in the TOPLUS tree.

For the kind of multicast trees we aim at building, each peer should be close to its parent in terms of network delay, while trying to join the multicast tree as high (close to the source) as possible. Each peer attempts at join time to minimize the number of hops from the source, and then the length of the last hop as much as possible. In the example of Figure 4.1, if p is a child of

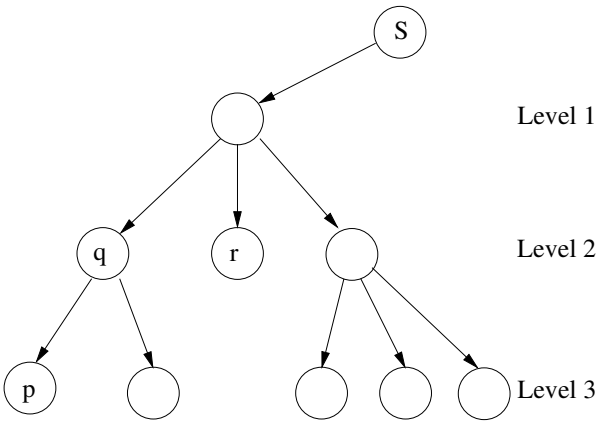


Figure 4.1: A simple multicast tree.

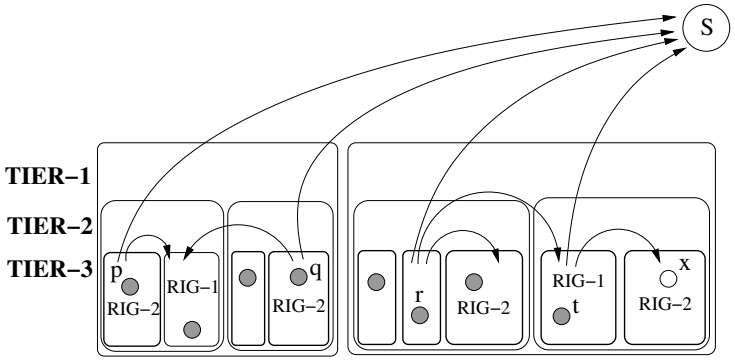
q and not of r, that is because p is closer to q than to r. Moreover, we want this condition to hold even if when p joins, q has not yet integrated the multicast tree: if q joins later and is closer to p than to r, then q should become a child of p. By trying to minimize the network delay for data transmission between peers at join time, we also avoid continuously rearranging peers inside the multicast tree, except when a peer fails or disconnects. For the kind of applications we are targeting the multicast trees should be as stable as possible. For a discussion on this topic, see Section 4.3.4 *Membership Management* below.

Although P2P networks are often characterized by a frequent joining and leaving of its members, we expect that peers connecting to a given multicast group are interested in remaining connected at least for the duration of the multicast session. This is our assumption for this chapter, see the next one for a discussion on MULTI+ introducing peer failures and sudden leaves. The TOPLUS network infrastructure offers a very interesting framework for the construction of such multicast trees.

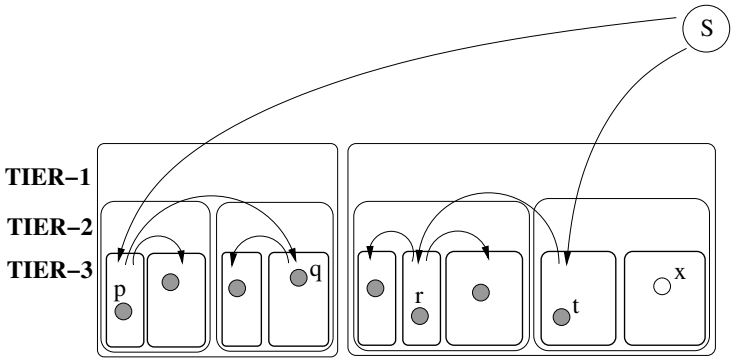
4.3.2 Building Multicast Trees

We use the TOPLUS network and look-up algorithm in order to build the multicast trees. Consider a multicast IP address m, and the corresponding key that, abusing the notation, we also denote m. Each tier-i group G_i is defined by an IP network prefix a_i/b where a_i is an IP address and b is the length of the prefix in bits. Let m_i be the key resulting from the substitution of the first b bits of m by those of a_i. The inner group that contains the peer responsible for m_i (obtained with a TOPLUS look-up) is the *responsible inner group*, or RIG, for m in G_i (note that this RIG is contained in G_i). Hereafter, we assume a single m, and for that m and a given peer p we denote the RIG in $H_i(p) \in$ tier-i simply as RIG-i of p. This RIG is a rendezvous point for all peers in $H_i(p)$. The deeper that a tier-i of a RIG-i is in the TOPLUS tree, the narrower the scope of the RIG as a rendezvous point (fewer peers can potentially use it).

In the simple 3-tier example of Figure 4.2(a), we have labeled the RIGs for a given multicast group (peers in gray are members of the multicast group), where all inner groups are at tier-3. The RIG-i of a peer can be found following the arrows. The arrows represent the process of asking the RIGs for a parent in the multicast tree. For example, p and q share the same RIG-1 because they are in the same tier-1 group. t's inner group is its RIG-1, but t would first contact a peer x (white) in its RIG-2 to ask for a parent (provided nobody is receiving the flow in its inner group). Note that this last peer is not in the multicast tree (Figure 4.2(b)). Some inner groups can be RIGs for more than one higher-tier group. If a peer is alone in its tier-1 group, it asks the source for a parent.



(a) The RIGs in a sample TOPLUS network.



(b) Sample multicast tree.

Figure 4.2: The hierarchical TOPLUS structure does not determine the multicast tree.

Assume a peer p in tier-$(i+1)$ (i.e., a peer whose inner group is at tier-$(i+1)$ of the TOPLUS tree) wants to join a multicast tree with multicast IP address m, which we call group m. The algorithm can be followed through the pseudo-code in Figures 4.3 and 4.4.

1. The peer p broadcasts a query to join group m inside its inner group (recall from Chapter 3 that in TOPLUS a peer knows all the members of its inner group). If there is a peer p' already part of group m, p connects to p' to receive the data (Figure 4.3, line 3).

2. If there is not such peer p', p must look for its RIG-i. A look-up of m_i inside p's tier-i group (thus among p's sibling groups at tier-$(i+1)$) locates the RIG-i responsible for m. p contacts any peer p_i in RIG-i (Figure 4.3, line 9), and asks for a peer in multicast group m (line 10). If peer p_i knows about a peer p'' that is part of m, it sends the IP address of p'' to p, and p connects to p''. Note that p'' is not necessarily a member of the RIG-i inner group. In any case p_i adds p to the list of peers listening to m, and shares this information with all peers in RIG-i. If p'' does not exist, p proceeds similarly for RIG-$(i-1)$: p looks up m_{i-1} inside p's tier-$(i-1)$ group (i.e., among p's sibling groups at tier-i). This process is repeated until a peer receiving m is found, or RIG-1 is reached. In the latter case, if there is still no peer listening to m, peer p must connect directly to the source of the multicast group.

```
function find_parent(m)
Require:
    m = multicast address
    p = peer looking for parent
 1: i ← p.tier
 2: p' ← NULL
    /* first try to find a connected peer p' in inner group */
 3: p' ← p.find_in_inner_group(m)
 4: while p' = NULL do
 5:     if i = 1 then
 6:         p' ← source_multicast
 7:     else
 8:         m_i ← p.key_prefix(m,i)
 9:         p_i ← p.look_up(m_i)
            /* p_i is responsible for m_i, thus belongs to RIG-i */
10:         p' ← p_i.ask_for_parent(m,p)
11:         i ← i − 1
12:     end if
13: end while
14: return p'
```

Figure 4.3: Obtaining a parent in the multicast tree.

One can see that the search for a peer to connect to is done bottom up.

Property 5 *When a peer p in tier-$(i+1)$ joins the multicast tree, by construction, from all the groups $H_{i+1}(p), H_i(p), \cdots, H_1(p)$ where p is contained, p connects to a peer $q \in H_k$ where $k = \max\{l = 1, \ldots, i+1 \,\exists r \in H_l$ and r is a peer already connected to the multicast tree$\}$.* That is, p connects to a peer in the deepest tier group that contains both p and a peer already connected to the multicast tree.

This process leads an arriving peer p to get as parent the peer q that shares with p the RIG at the lowest possible tier group containing both p and q. This assures that a new peer connects

function `ask_for_parent(m,p)`

Require:
m = multicast address
p = peer looking for parent
/∗ *peers* contains a FIFO *peers*$\{m\}$ for each multicast address m ∗/
1: $p' \leftarrow$ peers$\{m\}$.head
2: peers$\{m\}$.tail $\leftarrow p$
3: **return** p'

(a) Asking a peer in a RIG for a parent.

function `key_prefix(m,i)`

Require:
m = multicast address
i = tier of p's current look-up
/∗ from m, obtain key that maps to RIG-i when looked up ∗/
1: $pref_i \leftarrow p$.prefix_of_tier(i)
2: $l \leftarrow pref_i$.length
3: **return** ((m AND $(2^{32-l}-1)$) OR $pref_i$)

(b) Obtaining the modified key m_i that leads to RIG-i.

Figure 4.4: Auxiliary functions used in `find_parent`.

to the closest available peer in the network. Notice that even in the case of failure of a peer in a RIG-i, the information about the peers that have already visited the RIG is replicated in all other peers in the RIG-i. If a whole RIG-i group fails, while MULTI+ is undeniably affected, the look-up process can continue in RIG-$(i-1)$. We believe this property makes MULTI+ a resilient system, as will be shown later in Chapter 5.

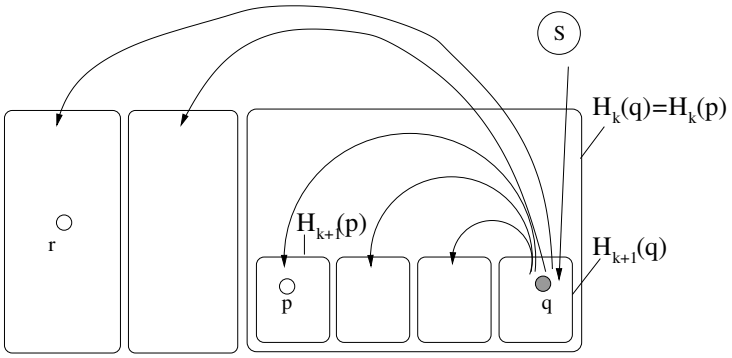
4.3.3 Multicast Tree Topology

Any peer in any group can become a node in a multicast tree; only the RIGs are fixed, for a given multicast group and a given TOPLUS network structure. The topology of the tree is thus not deterministic, and depends on the peers that want to join a given multicast tree. However, this tree is constrained to follow the TOPLUS structure. As it has been shown before, when a peer joins a multicast tree it will always connect to another peer that is close (according to TOPLUS proximity).

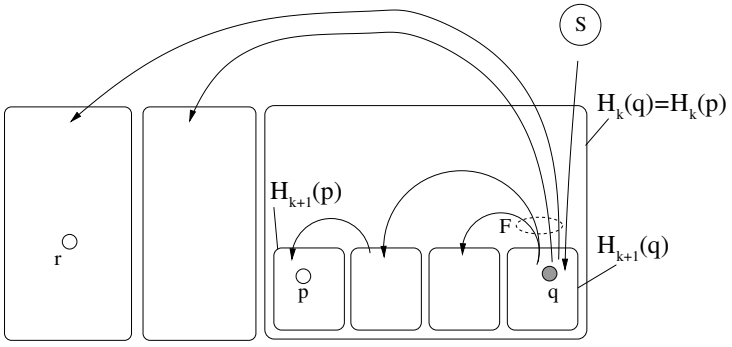
In general, we can assume that for any networked group in the Internet it is better to keep as much traffic as possible *inside* the group and avoid outbound traffic: Between ASes, routing at peering interfaces is done based mainly on economical considerations, whereas inside an AS normally other policies stand, more oriented to efficient networking. In most LANs, the outbound link to the Internet has an order of magnitude less bandwidth than the network itself. We show here that multicast on TOPLUS minimizes internetwork traffic.

Property 6 *For each group defined by an IP network prefix containing at least one peer connected to the multicast tree, there is only one inbound data flow.*

Proof: We proof by induction for a peer p in tier-$(i+1)$ contained in a set of groups (or networks defined by IP network prefixes) $H_{i+1}(p), H_i(p), \cdots, H_1(p)$:



(a) Inbound and outbound flows at each network.



(b) Inbound and outbound flows at each network, with $F = 4$ connections per peer.

Figure 4.5: Scheme showing flows going into and coming out of a group.

- If there is one peer q already connected to the multicast tree in p's inner group H_{i+1}, p connects to q. Else, p must connect to another peer outside H_{i+1}. Thus only one flow may go into the network H_{i+1}.
- In the look-up of a peer already connected to the multicast tree, assume p gets to $H_k(p)$ in tier-k. Thus, by property 5 no peer exists connected to the multicast tree in $H_{i+1}(p), H_i(p), \cdots, H_{k+1}(p)$. Only the data flow to p will go into $H_{i+1}(p), H_i(p), \cdots, H_{k+1}(p)$. If there is one peer $q \in H_k(p)$ receiving the multicast flow, p connects to q, and p adds no data flow into $H_k(p)$. Else p proceeds the look-up in $H_{k-1}(p)$. In any case only one data flow can go into the network defined by $H_k(p)$. □

Property 7 *For each group defined by an IP network prefix containing at least one peer connected to the multicast tree, the number of outbound data flows in the worst possible case is bounded by a constant.*

Proof: Assume as before a peer p in tier-$(i+1)$ contained in a set of groups (i.e., networks defined by IP network prefixes) $H_{i+1}(p), H_i(p), \cdots, H_1(p)$. Assume peer p is looking for a peer

to connect to. Without loss of generality, let $H_k(p)$ be the tier-k group where the RIG-k knows about a peer q connected to the multicast tree. p connects to q and adds an outbound data flow (from q to p) to $H_{k+1}(q)$. From the previous property, there can be at most one inbound data flow in $H_{k+1}(p)$. The same is valid for every sibling group of $H_{k+1}(q)$. The number of siblings of $H_{k+1}(q)$ is a bounded by a constant (which depends on the topology of the TOPLUS tree and the available routing information). Thus the number of outbound data flows from $H_{k+1}(q)$ towards tier-$(k+1)$ groups is a constant. For a worst case scenario, assume now that q is also the peer known by RIG-$(k-1)$ in $H_{k-1}(q)$. Applying the same reasoning to the siblings of $H_k(q)$, we have another constant number of outbound flows from $H_{k+1}(q)$ towards tier-k groups. A constant number of outbound flows at every tier, and the number of tiers being bounded by a constant, the number of outbound data flows is bounded by a constant for group $H_{k+1}(q)$. □

Property 8 *Using multicast over TOPLUS, the total number of flows in and out of a group defined by an IP network prefix is bounded by a constant.*

The proof is trivial from the two previous properties. □

Note that these properties apply in principle to an ideal scenario where each peer can provide data flows to as many peers as necessary. In the real world, there is a bandwidth limitation. We study later a modification of the protocol that allows each peer in the multicast tree to set a maximum number of children, depending on its bandwidth. We assume that most peers have links to the Internet with sufficient bandwidth (at least a medium-range ADSL).

4.3.4 Membership Management

Each peer p knows its parent q in the multicast tree, because there is a direct connection between them. Because p knows the RIG where it got its parent's address, if p's parent q in level i of the multicast tree fails or disconnects, p directly goes to the same RIG and asks for a new parent. If there is none, p becomes the new tree node at level i, replacing q. Then p must find a parent in level $i-1$ of the multicast tree, through a join process starting at said RIG. If p had any siblings under its former parent, those siblings will find p as the new parent when they proceed like p. If more than one peer concurrently tries to become the new node at level i, peers in the RIG must consensually decide on one of them. It is not critical if a set of peers sharing a parent q are divided in two subsets with different parents upon q's departure.

Join and leave is a frequent process in a P2P network, but we expect the churn to be rather low due to the fact that in a multicast tree, all peers seek the same content concurrently, throughout the duration of the session. Thus in this first chapter on MULTI+ we do not consider peers that connect and disconnect soon afterwards, which introduce high dynamics in the overlay network.

4.3.5 Limited Bandwidth in MULTI+

Depending on the structure of the TOPLUS network, a node in a multicast tree may have many descendants. For example, in a tier-1 group with 100 tier-2 subgroups, the level 1 parent may have up to 100 descendants. The bandwidth of a single peer may not be sufficient. We now introduce two load balancing techniques to deal with that issue:

1. In the protocol we have just presented, all peers arriving to a RIG-k looking for a parent in the multicast tree get assigned the same parent. This parent peer may become overloaded if a large number of children get the multicast flow from him. Assume that each peer can afford a maximum of F outgoing flows. For each set of F new peers arriving to a RIG-k, one of them should become the new parent for the next F peers to come later. Notice that this policy alone would increment the number of levels in the multicast tree, and consequently the delay from root to leaves. To see this, in Figure 4.5(b) only the first F groups can connect directly to q. The next F peers must connect to one of the first F. The height of the multicast tree is increased by one level for them (compare the hops from p to q in Figures 4.5(a) and 4.5(b)). The problem is that the RIG-k is only allowing F connections to q, but there may be other peers in $H_{k+1}(q)$ that can accept connections. After all the connections that q is able to serve have been occupied by its children in the multicast tree, q can suggest as a response a connection to another peer in $H_{k+1}(q)$ receiving the data flow. The height of the tree is increased, but the root-to-leaf delay does not augment much in general, because the suggested parent in $H_{k+1}(q)$ will be close to q, and it probably is receiving the data from q itself. Then, only if there are no available peers in $H_{k+1}(q)$, q can suggest a peer in one of the sibling groups of $H_{k+1}(q)$ in tier-$(k+1)$. There is another constrain though: independently of the peers willing to serve content in $H_{k+1}(q)$, the network outgoing link may not have enough bandwidth, and another sibling group may need to be used. Depending on whether the RIG or the parent peer manages new connections, some advantages and drawbacks appear.

2. At RIG-k, we assign to a new peer a parent that depends on the RIG-$(k+1)$ of the new peer. Taking Figure 4.5(b) as an example, peer r whose RIG-$(k+1)$ is not contained in $H_k(q)$ should have priority to connect directly to q. The reason for this is to allow peers from further groups to connect to a parent through fewer hops. Peers closer to q, that is, those contained in $H_k(q)$, can afford more hops when connecting to q through other peers in $H_k(q)$, because the delay is smaller. Thus root-to-leaf delays become more balanced over the multicast tree.

The properties presented previously are still valid with these policies. To understand why, consider that the number of inbound flows does not change and that the property for the outbound flows is based on the worst possible case: these policies just move part of the outbound flows from the most loaded group to its siblings. Moreover, note that this only happens in the case where the outbound link of the network of the worst case group cannot sustain the required flow of data. Thus we still have a constant bound in the number of flows coming in and out of a group.

We have considered so far a source capable of serving content to at least one peer in each tier-1 group in the TOPLUS tree. For modified TOPLUS trees with 8-bit prefix groups only in tier-1, this sums up to 256 potential flows. Although 256 flows may be perfectly affordable to commercial or institutional multicast (think of radio stations on the web today), it does not allow end users in the Internet to serve content which may potentially be of interest to many.

The problem is that our method to find a nearby existing peer is not valid between tier-1 groups. In the absence of a peer to connect to, we connect by default to the source. We propose another method here: the source accepts a limited number of direct connections. Once the limit is reached, new incoming connections receive a list of peers currently receiving the flow. The peers can then connect to the closest non-overloaded in the list (according to some easily measurable metric like RTT, or obtaining their distance from a pre-calculated coordinate space, as we shall see below). This process can equally apply to normal peers, when the number of connections is limited: a peer refusing a new connection must provide a list of peers already connected to him. To detect uncooperative or malicious behavior we can verify that those peers are indeed connected to the refusing peer. This process (challenge) is repeated until a valid peer is found. Note that collusion is not possible without introducing loops in the tree. A very powerful cheater might avoid having to serve content if he can fake other peers' IP addresses to answer to the challenge, so that those peers seem to be receiving data from the cheater. However, those peers should be normally reachable for other peers to connect to them (bypassing the cheater this time). A possible solution may be to refuse a connection request without the previous corresponding challenge. In general, if a directed acyclic graph (DAG) can be properly maintained, cooperation of peers is assured.

We can further discriminate among peers in a similar fashion as described above: Peers far from the source should have priority at getting direct connections, or even a pool of reserved bandwidth (e.g., reserving a fraction of the connections for peers farther than 200 ms). For peers close to the source, it is less critical to get the content directly, as long as at least one peer can get it from a peer near the source.

4.3.6 Parent Selection Algorithms

From the ideas exposed before, we retain two main parent selection algorithms for testing the construction of multicast trees.

- FIFO, where a peer joins the multicast tree at the first parent found with a free connection. When a peer asks a RIG (in fact, said peer is asking a given peer in the RIG, but we think that treating the RIG as an entity is not confusing in this context) for a parent, the RIG answers with a list of already connected peers. This list is ordered by arrival time to the RIG. Obviously, the first peer to arrive to a RIG connects closer to the source in the multicast tree. The arriving peer tests each possible parent in order from the first in the list until one that accepts a connection is found.

- Proximity-aware, where, *when the first parent in the list has all connections occupied*, a peer connects to the closest parent in the list still allowing one extra connection.

Note that MULTI+ does not always check if a peer connects to the closest parent in the list. The reason behind is that, while we implicitly trust MULTI+ to find a close parent, we prefer to connect to a peer higher in the multicast tree (fewer hops from the source) than to optimize the last hop delay. If MULTI+ works correctly, the difference between these two policies should not be excessive, because the topology-awareness is already embedded in the protocol through TOPLUS. We do not constantly rearrange the multicast tree structure with each arrival, because the longer a peer has been in the system the more likely it is to stay connected [74], and short-term delay minimization optimizations may not pay in the long run in terms of network stability.

4.4 Benchmarking MULTI+: Limitations of King

In this first experiment we test the characteristics of multicast trees built with MULTI+ using `king`. We use two sets of peers, one randomly (to some extent, as we will show later) generated, and the other formed by a subset of the hosts in the Planet-Lab project. For both of them we use `king` to measure the distance between arbitrary hosts, and the TOPLUS tree with exclusively 8-bit group prefixes in tier-1 we referred to as $16-bit+1$ regrouping in Chapter 3.

4.4.1 King Performance

We use `king` to measure the distance (i.e., latency) between two arbitrary hosts. First, `king` gives a good approximation of this distance in most cases, and second, `king` allows one to have an estimation of the distance between two hosts when more accurate methods like `ping` do not work. To see why, we proceed with our first experiment. We take the 328 hosts in Planet-Lab (at the time of the test), and we measure the distance with both `ping` and `king` from a host at Institut Eurécom. Figure 4.6 shows the relative error in percentage of the distance measured by `king` to the actual distance, measured with ping. 80% of the measurements present less that 20% of relative error.

But more interesting, with `king` we were able to measure 249 distances to hosts, while ping was only able to reach 191 destinations (obviously Figure 4.6 compares the hosts (171) that both ping and `king` were able to reach). This can be due to a number of reasons, like security policies not allowing some ICMP traffic, hosts disconnected from the network or going under maintenance, etc. `king` is clearly a compromise between the total accuracy of ping and the convenience of a more reliable set of hosts used to obtain the measurements, the DNS servers. Furthermore, `king` can easily measure from a single point of the network a matrix of distances among a large set of hosts, a much more convenient approach than logging into each host to make the latency measurement to the other hosts with `ping`.

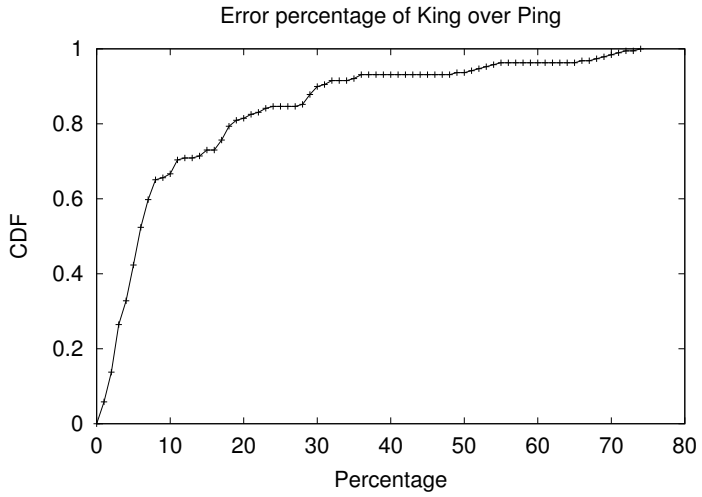


Figure 4.6: Error in `king` measurement compared to `ping` (from Institut Eurécom).

`king` measurement method is based on information contained in requests to DNS servers, thus is an *application level* method. `king` does not induce high stress in the network, like `traceroute` does, but it is clearly more expensive than an ICMP packet. Measuring the distance between two hosts can take several seconds. The population in our experiments is constrained by the time we can spend measuring the inter-host distance matrix. Even under the assumption that the matrix is symmetric, the time cost of the experiment is still $O(N^2)$ for N different hosts. To give an idea, a 300 peer set experiment, dedicating an average of 6 seconds to each measurement, takes almost 3 days to complete.

4.4.2 MULTI+ Performance

We measure how MULTI+ performs in two scenarios. The first one is a randomly generated set of peers, and the second is a set of Planet-Lab hosts. In both tests we measure two different parent selection policies when building the multicast tree: FIFO and proximity-aware. In all cases we test MULTI+ when we do not limit the maximum number of connections a peer can accept, and for a limited number of connections, from 2 to 8 per peer.

We measure the following parameters:

- The percentage of peers that connect to the closest peer in the system when they arrive, and the percentage of peers that are connected to the closest peer once the multicast tree is fully built (Tables 4.1, 4.2, 4.3 and 4.4).
- The percentage of the peers present at arrival time that are closer to a newly arriving peer p than the parent to which p actually connects to (Figures 4.7 and 4.10); the percentage

of the peers in the total system, when the full multicast tree is built, closer to a given peer than this peer's parent. Those figures *exclude* the peers that are actually connected to the closest peer (that is, those which have 0% of peers closer than their own parent in the multicast tree), and those connected to the source. They measure how bad is the connection made by MULTI+ when this decision is not optimal (Figures 4.8 and 4.11).

- The fan-out or out-degree of peers in the multicast tree. Obviously leafs in the tree have a fan-out of zero.

- The level peers occupy in the multicast tree(Figures 4.9 and 4.12). The more levels in the multicast tree, the more delay we incur along the transmission path and the more subject to errors due to peer failure the transmission becomes. There may be cases in the Internet where a routing going through more hops results in less latency than another, apparently more direct path, as shown in [122]. In any case, the latency through the path resulting of adding a hop to a given path is always larger than the latency through the original path.

Random peer set

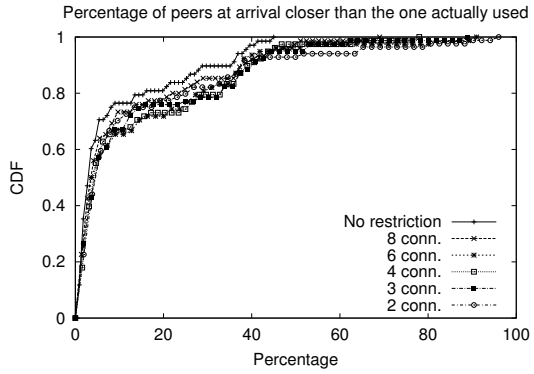
The set of IP addresses used in this scenario is not purely random. There are two reasons for this: one, we cannot measure the distances in an arbitrarily large set of peers. As explained before, measuring the whole matrix of distances for sets exceeding the 300 peers can represent weeks of work. Two, with a reduced set of peers, if we allow true randomness, we may finish with hosts exclusively located in tier-1 groups, a scenario for which MULTI+ is not conceived (and would be of no practical use).

Thus, we choose a set of tier-1 groups, and we randomly generate IP addresses *inside* those tier-1 groups. The tier-1 groups are chosen so that we can find at least one other tier-2 group inside.

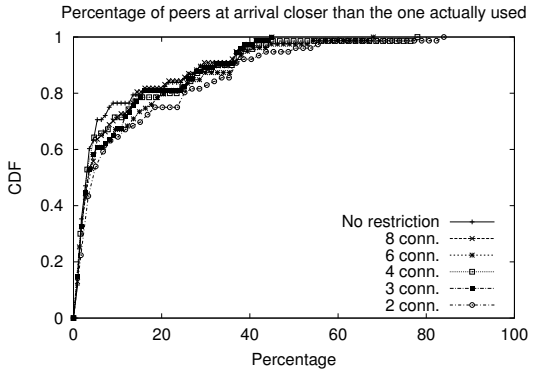
The high-level characteristics of the TOPLUS tree obtained are as follows: 19 tier-1 groups, 45 inner-groups and 200 peers.

	Fan-out limit							
	2	3	4	5	6	7	8	∞
to closest peer upon arrival	24	27	28	29	29	29	29	30
to closest peer in full system	9	10	9	12	11	11	11	12

Table 4.1: Random peer set, FIFO parent selection: Percentage of peers connected to closest peer, depending on the maximum number of connection allowed.

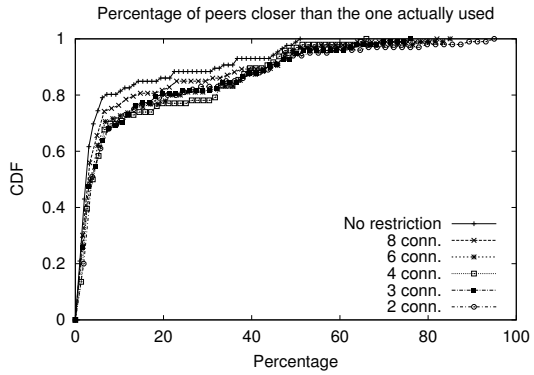


(a) FIFO parent choice.

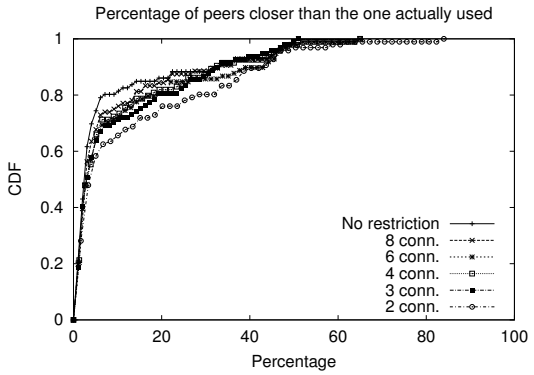


(b) Proximity-aware parent choice.

Figure 4.7: Random Peer Set: Percentage of peers found upon arrival closer than the one actually used (for those not connected to the source and not connected to the closest peer).

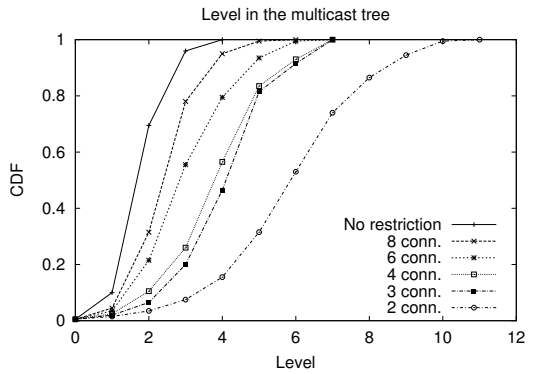


(a) FIFO parent choice.

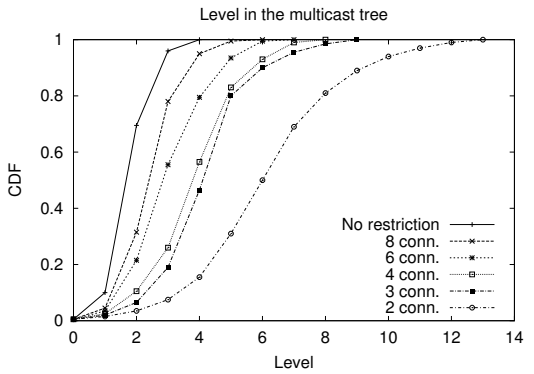


(b) Proximity-aware parent choice.

Figure 4.8: Random Peer Set: Percentage of peers found in the full system closer than the one actually used (for those not connected to the source and not connected to the closest peer).



(a) FIFO parent choice.



(b) Proximity-aware parent choice.

Figure 4.9: Random Peer Set: Level of peers in the multicast tree.

	Fan-out limit							
	2	3	4	5	6	7	8	∞
to closest peer upon arrival	35	36	40	39	31	32	29	31
to closest peer in full system	18	16	20	18	15	14	12	12

Table 4.2: Random peer set, Proximity-aware parent selection: Percentage of peers connected to closest peer, depending on the maximum number of connection allowed.

Planet-Lab peer set

In this scenario we have chosen the hosts in Planet-Lab as peers. There are a number of features that make this set interesting: they are real machines connected today to the Internet; they are more or less distributed around the world (many in the US, but so are most of the IP addresses); and they form a quite open platform, mainly contributed by universities and first-tier industrial partners. The risk of using a randomly chosen IP address set of small size is that it may introduce a scarcely representative but very negative bias in the experiment, by including a disproportionate number of addresses from obscure, badly connected ISPs.

The high-level characteristics of the TOPLUS tree obtained are as follows: 30 tier-1 groups, 61 inner-groups and 171 peers.

	Fan-out limit							
	2	3	4	5	6	7	8	∞
to closest peer upon arrival	37	38	38	35	36	37	36	42
to closest peer in full system	27	25	25	25	26	26	25	20

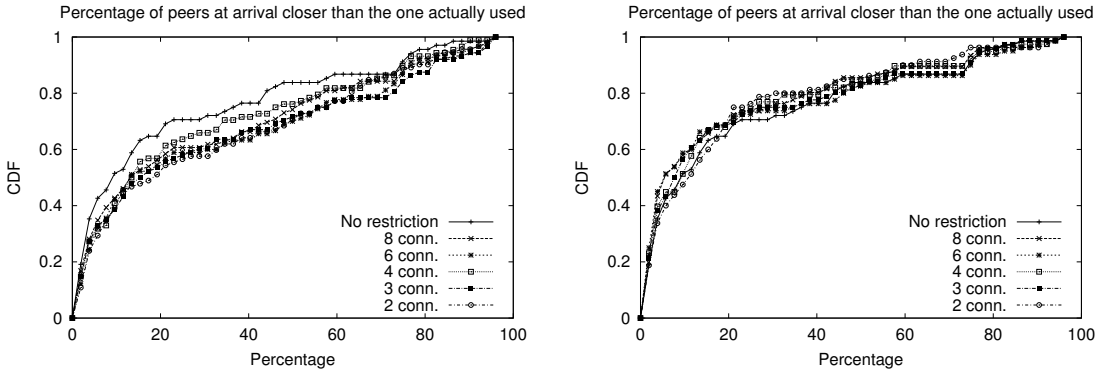
Table 4.3: Planet-Lab peer set, FIFO parent selection: Percentage of peers connected to closest peer, depending on the maximum number of connection allowed.

	Fan-out limit							
	2	3	4	5	6	7	8	∞
to closest peer upon arrival	46	47	45	45	42	43	46	42
to closest peer in full system	30	29	28	27	27	28	29	30

Table 4.4: Planet-Lab peer set, Proximity-aware parent selection: Percentage of peers connected to closest peer, depending on the maximum number of connection allowed.

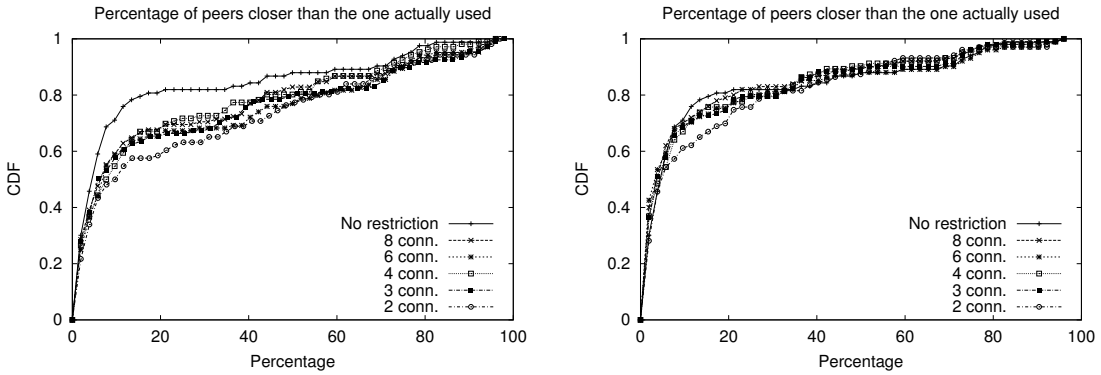
4.4.3 Observations

- In all experiments, when there is no restriction in the number of connections, the source receives as many connections as tier-1 groups are in the system.



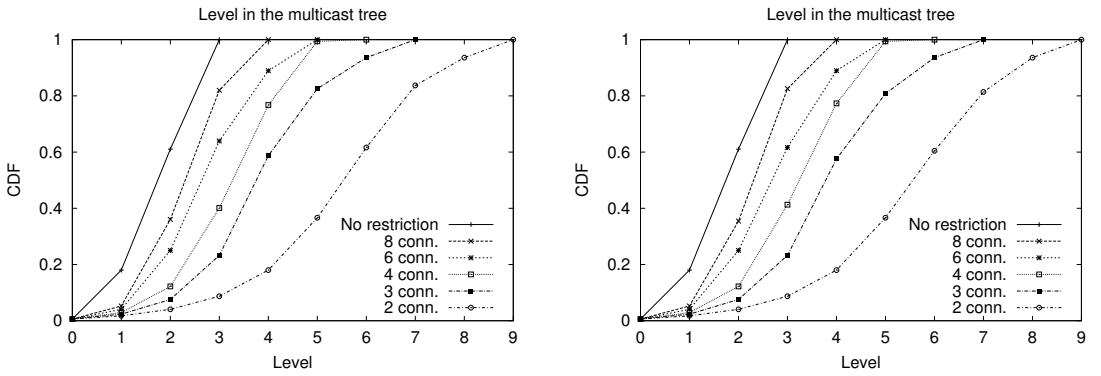
(a) FIFO parent choice. (b) Proximity-aware parent choice.

Figure 4.10: Planet-Lab Peer Set: Percentage of peers found upon arrival closer than the one actually used (for those not connected to the source and not connected to the closest peer).



(a) FIFO parent choice. (b) Proximity-aware parent choice.

Figure 4.11: Planet-Lab Peer Set: Percentage of peers found in the full system closer than the one actually used (for those not connected to the source and not connected to the closest peer).



(a) FIFO parent choice. (b) Proximity-aware parent choice.

Figure 4.12: Planet-Lab Peer Set: Level of peers in the multicast tree.

- With no connection limits, most peers are organized in a 3-level multicast tree (Figures 4.9(a) and 4.12(a)), which corresponds to most peers being in tier-2 groups in the TOPLUS hierarchy. However, a 4 connections limit per peer allows for a 5-level multicast tree. These figures may correspond to a scenario with peers connecting through 512 Kbps up-link DSL lines, and every peer feeding 128 Kbps data flows to each of its four children. This infrastructure would allow for multicasting medium-quality audio encoded with MPEG-1 Layer III. For the 2 connection limit case, the random set tree goes a little out of control with 13 levels, but the Planet-lab experiment achieves 9 levels, actually close to $\log_2(171) = 7.42$.

- Among the non-optimal connections in the Planet-Lab tree, the error (Figure 4.10(a)) can go further than in the random set tree (Figure 4.7(a)), but the number of optimally connected peers is greater in the Planet-Lab tree (Tables 4.3, 4.4) than in the random set (Tables 4.1, 4.2).

- The careful reader may have observed that in some cases, allowing for more connections per peer decreases the number of peers that connect to the closest parent in the multicast tree (for instance, see Tables 4.2 and 4.4). Indeed, in Table 4.2, we observe that 4 connections per peer make 40% of the peers connect to the closest parent, while an unrestricted number of connections per peer results in only 31% of the peers in the same situation. This is easily explained. One must not forget that the proximity of a peer to its parent is an indication of efficiency when, constrained by a small number of connections per peer, we are forced to build very deep trees to fit all the peers in the multicast infrastructure. Proximity assures that the end-to-end delay will not grow without control. However, if bandwidth is not a problem, and one peer can send data to as many peers as desired, it is better to connect as close to the source as possible. With no restrictions whatsoever, *all* peers would connect to the source, and 0% of the peers would connect to the closest parent, but the situation would be far from a disaster!

- The proximity-awareness improves the number of optimal connections both in the Planet-Lab set (Tables 4.3, 4.4) and the random set (Tables 4.1, 4.2). However, the error in the non-optimal connections is also reduced in the Planet-Lab set with proximity-aware parent selection algorithm (Figure 4.10(b)) while the Random peer set (Figure 4.7) does not benefit significantly from the measurements made to select the closest parent. Still, the difference between the proximity-aware and FIFO algorithms is small enough to conclude that MULTI+ itself creates fairly good topology-aware multicast trees. This is strengthened by the fact that limiting the connections does not affect a lot the optimality of the generated tree.

4.5 Benchmarking MULTI+: Simulation in a Coordinate Space

Obviously, the $O(N^2)$ cost of actively measuring the full inter-host distance matrix for N peers limits the size of the peer sets we can use. P2P systems must be designed to be potentially

very large, and experiments should reflect this property by using significant peer populations. Some methods ([82, 123, 124, 125]) to map hosts into a M-dimensional coordinate space have been developed. Each host is then represented by a point $x_i = (x_{i1}, \ldots, x_{iM})$ in the coordinate space. The main advantage is that given a list of N hosts, the coordinates for all of them can be actively measured in $O(MN)$ time (the distances of the hosts to a set of M *landmark* hosts, with $M \ll N$). Then, the inter-host distance matrix can be calculated off-line, and the population of our experiment is only limited by CPU power and memory (a far less tight bound).

4.5.1 TC Coordinates

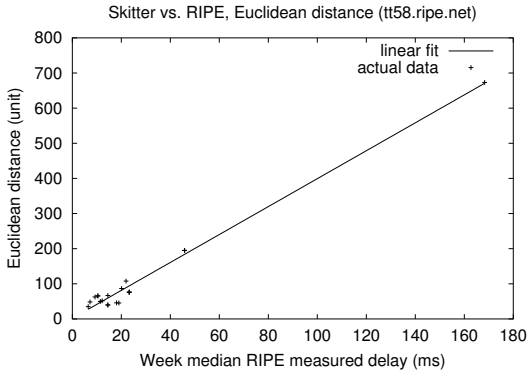
CAIDA [126] offers to researchers a set of network distance measurements from so-called *Skitter* hosts to a large number of destinations. `skitter` is a traffic measurement application developed by CAIDA. In a recent paper [125], the authors have used the Skitter data and some more distance measurement data to obtain a multi-dimensional coordinate space representing the Internet. A host location is denoted by a point in the coordinate space, and the latency between two hosts can be calculated as the distance between their corresponding points. The authors of [125] have kindly provided us with the coordinates of $196,297$ IP addresses for our study. Hereafter we call these IP addresses the TC (after the authors Tang and Crovella) address set, and their set of coordinates the TC coordinate space. To validate the accuracy of the Skitter data we have, we measured the degree of correlation between distances calculated from this coordinate space and the real measured distances for some random samples.

4.5.2 Performance of TC Coordinate Space

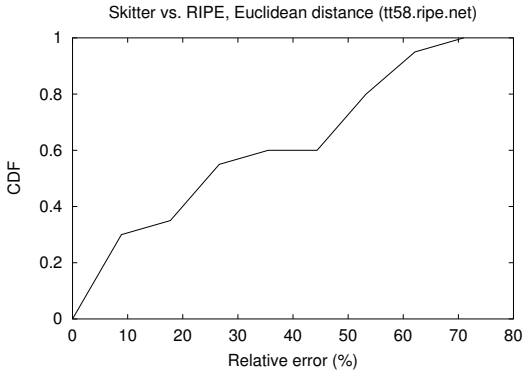
RIPE [96] has deployed a number of machines (probes) that measure the delay among each other. We use the median value of each of these delays measured during a week as the distance between their two corresponding machines. We could find one RIPE TTM (Test Traffic Measurements) probe in the TC address set. We found another 15 probes in the 26-bit prefix neighborhood of some IP address in the TC address set. We assume that the delay for the probes is similar to the IP addresses in the same 26-bit prefix, because each of those prefixes represents a network of 64 machines. In a M-dimensional space, Euclidean distance d^E between any two hosts identified by their corresponding points x_i and x_j is defined as:

$$d^E(x_i, x_j) = \sqrt{\sum_{k=1,\ldots,M} (x_{ik} - x_{jk})^2}$$

We obtain encouraging results, as can be seen in Figure 4.5.2. The authors of [127] claim that metrics other than the Euclidean yield better results, but we did not observe noticeable differences, so we use the Euclidean distance hereafter. We also find a good correlation between distances measured with `ping` and `king` [99] on different sets of hosts of the TC address set and



(a) Linear fit between calculated and measured distances.



(b) Relative error to linear fit data.

Figure 4.13: Euclidean distances compared to measured delay.

those calculated with the TC coordinates (we refer the interested reader to Appendix A for detailed results). In any case, the RIPE probes should give precise and stable measurements, while other sets may present less accurate results.

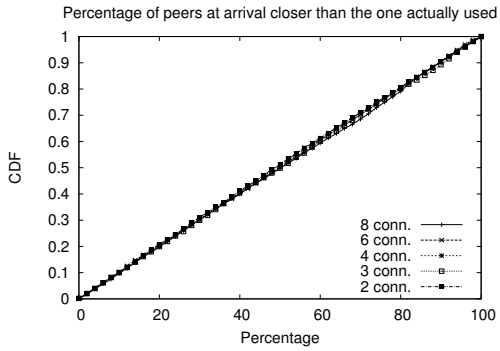
4.5.3 Multicast Tree of Five Thousand Peers

In this experiment we test the characteristics of multicast trees built with MULTI+ using a set of 5,000 peers from the TC address space. We use the TC coordinate space to measure the distance between every pair of hosts. In order to make the experiment as realistic as possible, we use a TOPLUS tree with routing tables of reduced size (thus introducing a distortion in the topological fidelity of the resulting tree, as seen in Section 3.5.1). The 5,000 peers are organized into a TOPLUS tree with 59 tier-1 groups, 2,562 inner-groups, and up to 4 levels. We evaluate the two different parent selection policies described before: FIFO and proximity-aware. We also compare these two approaches with random parent selection. In all cases we test MULTI+ when we do not set a limit on the maximum number of connections a peer can accept, and for a limited number of connections, from 2 to 8 per peer. This limitation also applies to the source. However, in our test we give the source a non-existent IP address, so that our measurements do not get biased by the choice of a given peer. Thus we consider exclusively the latency introduced by the transmission of the data through the MULTI+ tree, and the peers connected directly to the source get 0 latency. In the test we measure the following parameters, presented here using their CDF (Cumulative Distribution Function):

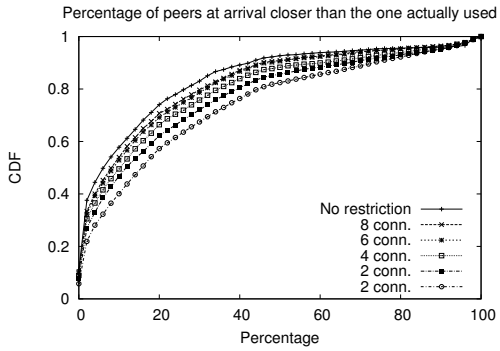
- The percentage of peers that connect to the closest peer in the system when they arrive (Figure 4.14).
- The percentage of the peers in the total system, when the full multicast tree is built, closer to one peer that this peer's parent. Those figures *exclude* the peers directly connected to the source (Figure 4.15).
- The fan-out or out-degree of peers in the multicast tree. Obviously leafs in the tree have 0 fan-out.
- The level peers occupy in the multicast tree (Figure 4.16). The more levels in the multicast tree, the more delay we typically incur in along the transmission path and the more the transmission becomes subject to losses due to peer failure. See however [122].
- The latency from the root of the multicast tree to each receiving peer (Figure 4.17).
- The number of multicast flows that go into and out of each TOPLUS group (network) (Figure 4.18).

There are two points that are capital in order to properly evaluate the results of these experiments, specially when compared with the reduced peer sets presented in Section 4.4:

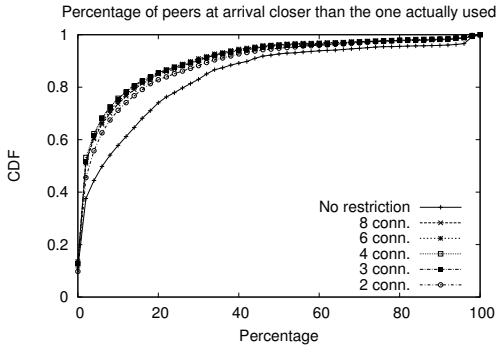
- Many peers (31%) are *alone* in their respective inner groups. Moreover, 65% of the peers are in inner groups with 3 or fewer peers. This reinforces the impact of the "topology-awareness" introduced by the TOPLUS layout, because connections inside an inner group are rare, and therefore mainly made among *different* inner groups.



(a) Random parent selection.



(b) FIFO parent selection.



(c) Proximity-aware parent selection.

Figure 4.14: Percentage of peers found upon arrival closer than the one actually used (for those not connected to the source).

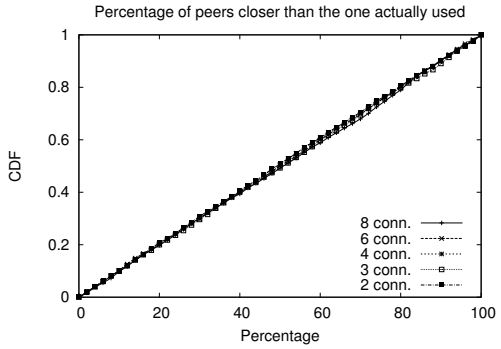
- Peers join the system in a completely *random order*. This means that close peers do not arrive in sequence. In particular, when a peer appears in an inner group where already a peer exists, the new peer may not find an available connection in the existing peer. To address that problem, one can argue that a peer should reserve a few connections for peers from the same inner group, or drop one connection in favor of the newcomer from the same inner group. As we have discussed before, a multicast source may use this kind of policy to assure maximal dissemination over the whole Internet with limited resources. However, we do not proceed like that, for a number of reasons: first, one cannot tell *a priori* whether peers will appear in a given inner group, and reservation would be a waste of resources; second, given the nature of P2P systems (hosts in edge networks of the Internet), bandwidth is a scarce enough resource itself, especially upstream; and third, given that peers' connections may be highly volatile, and that the longer a peer has been in the system the more likely it is to stay connected [74], short-term delay minimization optimizations may not pay in the long run in terms of network stability.

	Fan-out limit							
	2	3	4	5	6	7	8	∞
to closest peer upon arrival	5.9	7.9	9.0	9.7	9.8	9.9	10.3	11.1
to closest peer in full system	3.7	5.4	5.9	6.3	6.4	6.6	6.8	7.4

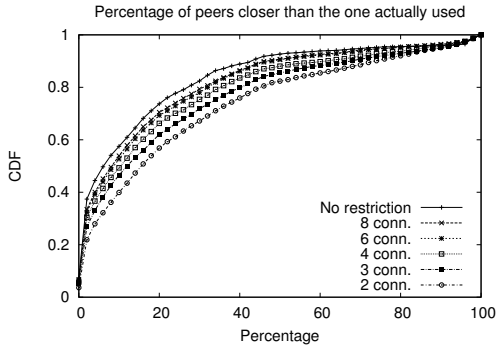
Table 4.5: FIFO parent selection: Percentage of peers connected to closest peer, depending on the maximum number of connection allowed.

	Fan-out limit							
	2	3	4	5	6	7	8	∞
to closest peer upon arrival	9.9	12.9	13.0	13.7	13.7	13.3	12.7	11.1
to closest peer in full system	6.3	8.1	7.8	8.5	8.3	8.2	7.9	7.4

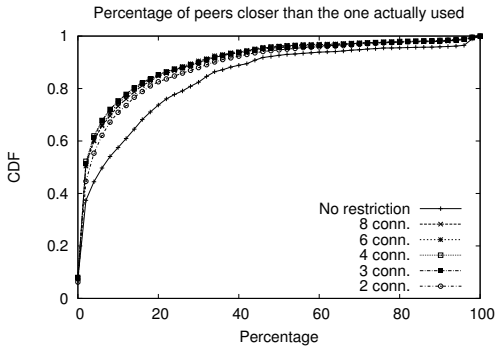
Table 4.6: Proximity-aware parent selection: Percentage of peers connected to closest peer, depending on the maximum number of connection allowed.



(a) Random parent selection.

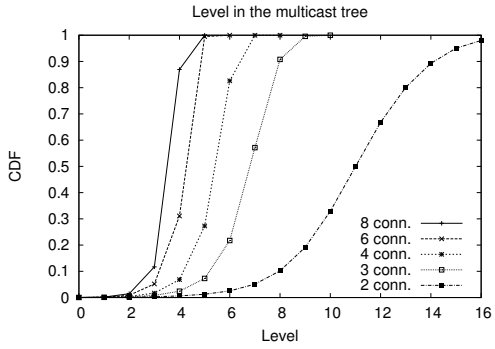


(b) FIFO parent selection.

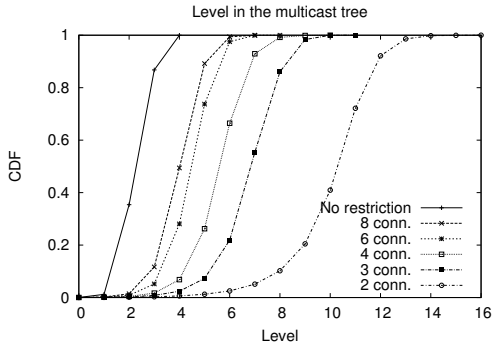


(c) Proximity-aware parent selection.

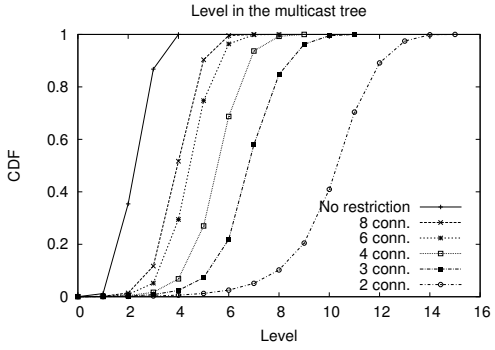
Figure 4.15: Percentage of peers in the whole system closer than the one actually used (for those not connected to the source).



(a) Random parent selection.

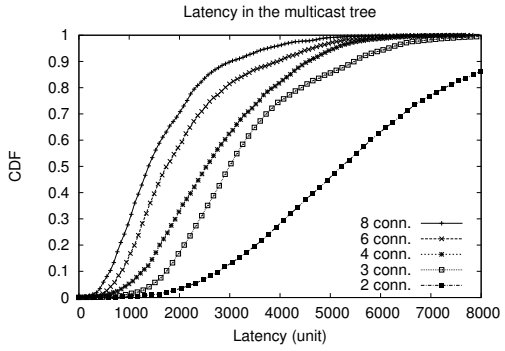


(b) FIFO parent selection.

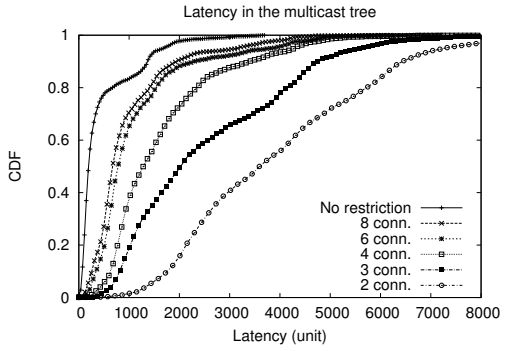


(c) Proximity-aware parent selection.

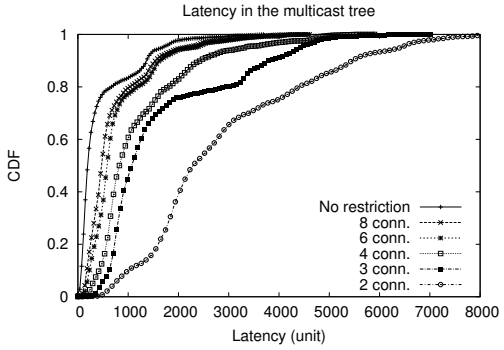
Figure 4.16: Level of peers in the multicast tree.



(a) Random parent selection.

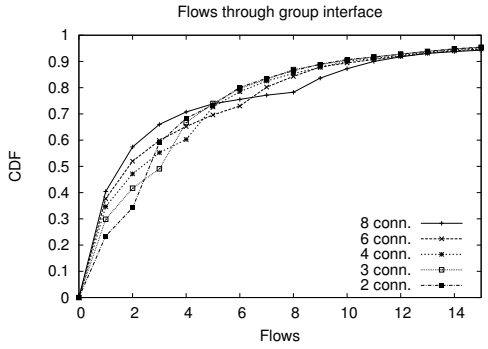


(b) FIFO parent selection.

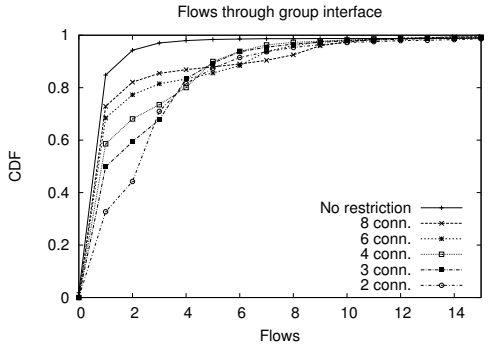


(c) Proximity-aware parent selection.

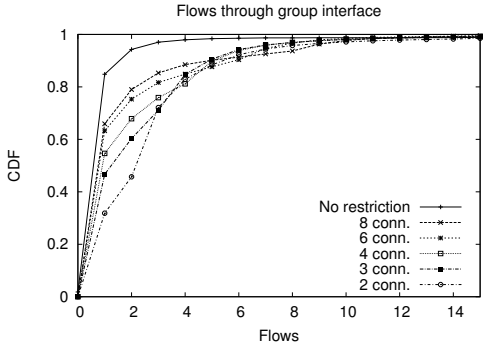
Figure 4.17: Latency from root to leaf (in TC coordinate units) in the multicast tree.



(a) Random parent selection.



(b) FIFO parent selection.



(c) Proximity-aware parent selection.

Figure 4.18: Number of flows through group interface.

From our experiments we obtain very satisfactory results. From Figures 4.14 to 4.18 we draw a number of conclusions:

- We do not need a large number of connections to benefit from MULTI+ properties: 3 connections are sufficient and in practice affordable for broadband users; the marginal improvement of 8 connections is not very important.
- The proximity-aware policy performs better than FIFO in terms of latency (Figure 4.17) and last-hop optimization (Figure 4.15). However, in terms of the number of flows per group (Figure 4.18) and the distribution of peers across levels of the multicast tree (Figure 4.16), they are very similar. That is because both trees follow the TOPLUS structure, but the proximity-aware policy takes better decisions when the optimal parent peer has no available connections.
- In Figure 4.14(c) and 4.15(c) we can see that, for the proximity-aware policy, an unlimited fan-out configuration makes the last hop latency less optimal than the case where the fan-out is limited. This is normal, since when we have available connections, peers connect as high in the multicast tree as possible. See in Figure 4.16 how peers are organized in fewer levels, and in Figure 4.17 how the root-to-leaf latency is better for the unrestricted connection scheme. Still, we can assert that the multicast tree is following (when possible) a topology-aware structure, because most peers connect to nearby parents. This effect has already been observed in Section 4.4.2.
- The random parent selection policy organizes the tree in fewer levels than the other two (Figure 4.16(a)), because connections are not constrained to follow the TOPLUS structure. However those connections are not optimized, and the resulting performance in any other aspect is considerably poorer.
- In Figure 4.17 a dip can be observed between 0.8 and 0.9 of the y axis, for almost any configuration when using MULTI+. One possible explanation is the presence of a few tier-1 (or deeper in the tree, but large in any case) groups containing many peers connected through low-latency links. If the peers inside the group are able to build a tree among themselves, the aggregated latency through them will be small, making for a rough 5% of peers with very similar latency to the root of the multicast tree.

We show in Table 4.7 the maximum number of flows going through a TOPLUS group interface. MULTI+ reduces the average and maximum number of flows *per network*, when compared to an arbitrary connection scheme and assuming each TOPLUS group (IP network prefix) represents a physical network. On average, each interface is used by 3 or 4 flows under MULTI+ (independently of the parent choice policy), while 7 are used with the random algorithm. The proximity-aware parent selection clearly improves, at the low extra cost of determining the closest parent, both the end-to-end latency (Table 4.8, 95% confidence intervals) and the number of flows per group (Table 4.7).

In this set of 5,000 peers, even limited to two connections per peer, MULTI+ builds multicast trees with significantly better properties than a random one. Indeed, MULTI+ is conceived for

	Fan-out limit				
	2	3	4	6	8
FIFO	260	198	145	107	109
Proximity-aware	165	109	101	39	55
Random	885	919	907	931	897

Table 4.7: Multicast Tree of 5,000 Peers: Maximum number of flows through a group interface.

	Fan-out limit				
	2	3	4	6	8
FIFO	$3,905(\pm 53)$	$2,527(\pm 45)$	$1,641(\pm 32)$	$1,135(\pm 28)$	$965(\pm 24)$
Proximity-aware	$2,871(\pm 47)$	$1,630(\pm 35)$	$1,205(\pm 26)$	$797(\pm 19)$	$698(\pm 18)$
Random	$5,339(\pm 60)$	$3,291(\pm 40)$	$2,742(\pm 35)$	$2,072(\pm 33)$	$1,646(\pm 28)$

Table 4.8: Multicast Tree of 5,000 Peers: Average latency from root to leaf (in TC coordinate units).

large scale deployments counting many peers, and its properties become asymptotically better the more peers we introduce in the system.

4.6 Conclusion

We have presented MULTI+, a method to build application-level multicast trees on P2P systems. MULTI+ relies on TOPLUS in order to find a proper parent for a peer in the multicast tree. MULTI+ exhibits the advantage of being able to create topology-aware content distribution infrastructures without introducing extra traffic for active measurement. Admittedly, out-of-band information regarding the TOPLUS routing tables must be calculated offline (a simple process) and downloaded (like many P2P systems today require to download a list of peers for the join process). This inconvenience is addressed in the next chapter. The proximity-aware scheme improves the end-to-end latency, and using host coordinates calculated offline and obtained at join time (as is done for TOPLUS) avoids all active measurement. MULTI+ also decreases the number of redundant flows that must traverse a given network, even when only few connections per peer are possible, which allows for better bandwidth utilization.

Chapter 5

Robustness of MULTI+ : Towards a Practical Implementation

Failure is the opportunity to begin again, more intelligently.

Henry Ford

MULTI+ is an application-level multicast protocol for content distribution over a peer-to-peer (P2P) TOPLUS-based network. TOPLUS organizes peers into hierarchical groups defined by IP network prefixes mainly obtained from BGP tables. In this chapter we present a study on the resiliency of the properties of multicast trees built with MULTI+ (low-delay and topology-awareness) when massive failure or disconnection of peers occur. Tree reconstruction introduces more hops in the peer-to-peer streams but succeeds at maintaining the end-to-end latency at similar values to those of the original overlay. At the expense of sacrificing a little accuracy on the topology-awareness of the multicast data flows, we propose a much more straightforward hierarchical partition of the IP address space. We study MULTI+ with this peer organization scheme and we obtain very satisfying results both in simulations with 5,000 peers using an Euclidean coordinate space and in a real scenario on a set of Planet-Lab hosts.

5.1 Introduction

We have based our P2P multicast protocol (MULTI+) on TOPLUS because of its inherent topology-awareness. TOPLUS derives the groups from the network prefixes contained in BGP tables or from other sources. MULTI+ benefits from TOPLUS properties to build low-latency multicast trees avoiding probing of links. In this chapter we present a simplified approach to TOPLUS that we call Hierarchical Internet Partition, avoiding the need to collect BGP infor-

mation, at the cost of losing some accuracy in topology-awareness.

We are not aware of practical research on how massive failure of peers affect the properties of overlay networks or the consequences of peers leaving those systems. The study of the eventual degradation is specially important in overlays that try to optimize some metric over its members. In this chapter we determine the robustness of MULTI+ by analyzing the effect of peer failure on the characteristics of the multicast trees. As MULTI+ tries to optimize end-to-end delay on the multicast trees at tree construction time, the sudden leave or failure of a significant fraction of peers severely alters the scenario where the trees are first built. By using the same algorithms that are used to initially build the trees, MULTI+ reconstructs the multicast structure, without reorganizing the full tree. Only the peers directly affected by a failure or unexpected leave relocate in the multicast tree. However, we will see that MULTI+ is able to maintain the properties of the multicast trees within acceptable bounds of the initial deployment.

For related work on application-level multicast we refer the reader to Section 4.2 in the previous chapter. As we are not aware of similar studies on the impact of peer failures on the characteristics of overlay networks, we do not provide a specific section for related work in this chapter. ZIGZAG [121] offers low root-to-leaf latency, but their study on the impact of peer failure does not cover how this latency is affected.

We organize this chapter in the following sections: In Section 5.2 we introduce a new metric we consider useful for multicast tree topology-awareness evaluation. Section 5.3 properly studies the resilience of MULTI+ against peer failure. Section 5.5 introduces the Hierarchical Internet Partition, a simplification of TOPLUS, and evaluates its performance. Section 5.6 presents some results obtained on a reduced set of peers on the Planet-Lab testbed. Finally, we conclude in Section 5.7.

5.2 New Metrics

In order to present the results of the evaluation of the robustness of MULTI+ under peer failure and to attempt to improve the characteristics of the multicast trees, we have developed two new metrics that we present in this section.

5.2.1 Parent Selection Algorithm

In the algorithm depicted in Figure 4.4(a), we assumed for the ideal case that a peer can receive as many connections as necessary. However, in reality, a peer may refuse a connection if a limit has been attained. We slightly modified the algorithm in Figure 4.4(a) to make it return a list of potential parents (peers that have joined the multicast tree before the arriving one). When a peer gets to a RIG to find a parent, the RIG answers with a list of already connected peers. We have

studied two different methods to select a parent from this list: connecting to the first parent still allowing a connection (FIFO) and connecting to the closest parent in the list when the first has reached its fan-out limit (proximity-aware).

In the proximity-aware parent selection algorithm the latency to each available parent needs to be known or measured. We can however try another approach, where a peer connects to the parent whose latency to the root of the tree plus the latency between the parent and the peer is minimized. Each peer still makes the same active measurement as before, and the latency to the root is calculated as the sum of the latency to the parent plus the latency from the parent to the root. The latency thus calculated is stored at each peer at join time. This recursive process gives the latency of each peer to the root with a single measurement. However, one needs to trust the information given by the parent: this last could announce a huge latency to the source in order to avoid having to feed data to other peers.

Our tests show that using this end-to-end latency method does not improve much the multicast tree construction: the performance obtained is similar to that where just the latency to the parent is used. There is still another drawback: when a peer fails and all its children must reconnect to other peers, the stored root-to-leaf latency must be updated for all the children of the peer that failed all the way down the multicast tree. Comparing these updates with the traffic due to streaming data in the overlay multicast tree, the former represent a minimal overhead, but still one not improving significantly the performance of the system.

5.2.2 Multicast Tree Optimality Metric

We present a metric to evaluate how optimal is the parent chosen by a given peer, compared with the other available possibilities. In general, a peer in MULTI+ seeks a parent that is (1) as close as possible to the source, in terms of overlay hops, and (2) as close as possible to said peer, in terms of latency. Given a peer p connected to a parent q, we try to find a parent q' closer to p, which is at least as close to the source in terms of hops in the multicast tree as q is, and has at least one empty connection available.

If we find such a q', we compare how much higher is the network latency from q to p than it is from q' to p. Given that function $d(p,q)$ returns the latency between peers p and q, this ratio R is calculated as follows :

$$R = \frac{d(p,q)}{d(p,q')} - 1$$

Thus, if we find no better parent than the current one, $q \equiv q'$ and this ratio r is 0.

5.3 MULTI+ under Peer Failure

In order to test the impact of peer failures on the performance of MULTI+, we must first create a multicast tree. We prefer to use measured data to build a realistic scenario instead of using a topology simulator. As we can not control how the algorithm is going to behave, we need a set of peers participating in the multicast session and the full mesh of latencies among them, because in principle a peer can connect to any other to receive the data.

We test the behavior of a MULTI+ tree when a number of peers fail (or leave without warning the others). In principle, a peer leaving should warn its children and parent of its leaving. The peer leaving also notifies the RIGs it visited when it joined the multicast group. In this manner, the peer leaving is erased from the list of peers in the multicast tree at each RIG.

The multicast service suffers from peer failure: The children of the peer leaving notice immediately that the flow of data has been interrupted. When they visit the corresponding RIG to get a new parent, they also submit the identity of the failed parent, so that lists in the RIGs can be updated. If a peer fails (or leaves without notice), its parent is unaffected. Even if the consequences of a peer leaving the system without honoring the corresponding protocol can be minimized to a temporal service interruption, it is preferable to achieve continuous data delivery through proper hand-over from parent to children. This requires the peer that leaves to keep on feeding data to its children until they have all found alternative parents.

We concentrate our study on how well MULTI+ maintains the characteristics of the multicast tree when peers fail. For this analysis we are using what we call a *Modified* TOPLUS tree (see Section 3.5.1), where we limit the routing tables' size by regrouping (thus *compacting*) the network prefixes under virtual groups (tier-1 prefixes not found in BGP tables). We discuss TOPLUS trees a little more in depth later in this chapter, but we refer to Chapter 3 for a complete background on TOPLUS.

We have already shown in Chapter 4 that MULTI+ can build static multicast trees with interesting properties. However, due to the nature of P2P systems, where the infrastructure is provided by the users of the service, the effects of leaving or failing multicast tree nodes must be specially considered. Our tests analyze a number of properties of the MULTI+ tree, after 5%, 10%, 20% and 40% of the peers leave improperly. 5-10% may correspond to failures due to final user or system (Home PC) error. 40% failure is the scenario of a massive attack against the content distribution network (DoS or virus), or maybe just people disconnecting from a deceptive movie transmission. In this last case we would expect users to properly disconnect from the system, though. For each case, we also plot the characteristics of the multicast tree when no errors happen, so that the reader can appreciate the good properties of MULTI+ and its degree of resilience to peer failures.

Through Figures 5.1 to 5.4 we plot the CDF (Cumulative Distribution Function) of the following properties:

- Ratio R of the latency to the parent in the multicast tree and to the most optimal peer (Figure 5.1), as defined in Equation 5.2.2.
- Distribution of peers across the levels of the multicast tree (Figure 5.2).
- Latency from root to leaf in the multicast tree (Figure 5.3).
- Number of flows through group interface (Figure 5.4).

We plot those properties when allowing an unlimited number of connections per peer, 8 connections per peer, or just 4. Although not realistic, the unrestricted number of connections represents for most properties of the MULTI+ tree a performance upper bound. The algorithm used for parent selection in all experiments is "proximity-aware", as it yielded the best results in the absence of peer failure.

We summarize our observations with the following remarks:

- We first observe that when there are no failures, the algorithm performs quite similarly when we do not limit the number of connections per peer or when we restrict the number to 8 or 4. More than 60% of the peers connect to the most optimal parent, that is, the closest peer higher in the multicast tree (closer to the source). The proximity of child to parent connection is preserved (slightly deteriorated) in the presence of errors (Figure 5.1). This is good, but not surprising, because orphan children use the same TOPLUS algorithm to look for a new parent. The degradation in the choice of a parent is more important the more we restrict the number of connections per peer. Indeed, when there is no restriction, the children of a failed parent peer can immediately connect to the grandparent peer.
- In Figure 5.2 we see that the level of peers in the multicast tree is preserved for a low error rate. However, large-scale failure pushes peers down the tree up to 34 levels, as we can see for the 4 connection tree and 40% peer failure in Figure 5.2(c). The fewer connections we allow per peer, the more important the effect for low failure rates (compare the 10% failure rate plot for an 8 connections limit in Figure 5.2(b) and the same failure rate with a limit of 4 connection per peer in Figure 5.2(c)). The unrestricted MULTI+ tree is unaffected, because it is always possible for all the children of a peer that fails to connect to their grandparent in the multicast tree (there are always available connections).

  This phenomenon is easily explained: The higher the failure rate, the higher the probability of failure of a peer close to the multicast source. The higher a peer is in the tree (closer to the source), the more difficult it is to find an available connection at the same level, and the more children this peer has below in the multicast tree (note that the number of nodes in each level of a tree increases exponentially with the level number). In the worst case, a peer directly connected to the source, when failing, forces its direct children (except the one that takes its place) to connect to leaf peers, low in the tree. Then the leaf peers under the direct children of the failed parent may find themselves twice as far from the source! This explains the degradation in the parent choice seen in Figure 5.2(b) and 5.2(c). This

approach may be considered inefficient, but this way only the direct children of a failed peer need to look for a new parent, instead of rebuilding the whole branch.

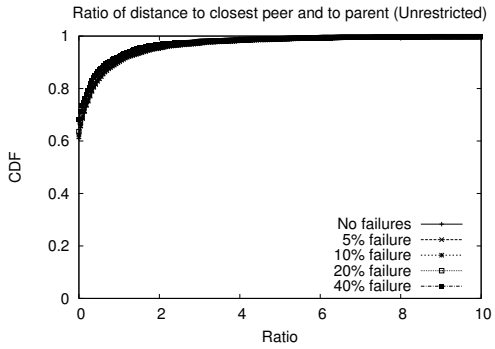
- From the previous point one would expect end-to-end latency to degenerate to unsustainable levels when peers fail. That would be indeed the case, if it was not for the topology-aware properties of the MULTI+ tree. See Figure 5.3. We see that increasing the proportion of failed peers does not affect the end-to-end latency a lot. The increased latency due to failed peers is more pronounced the tighter the restriction of the number of connections per peer. Fewer connections makes more difficult to find a suitable new parent for the orphan children of the failed peer. We further explore this level-latency relation next in Section 5.4.
- The number of flows per group is plotted in Figure 5.4. We observe that the utilization of each network prefix is low, just a few flows per group. This is due to the topology-aware algorithms we are using to build the multicast trees. The structure of these trees is mainly unaffected by the failure of peers, due to the constraints imposed by the TOPLUS tree. The number of flows per group is just a little reduced because there are fewer peers, thus fewer data flows. Notice that we do not present the groups that degenerate to 0 flows because all peers inside have failed.

5.4 Detailed Study of the Effect of Peer Failures

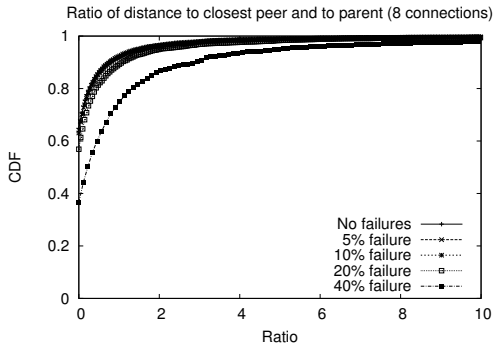
Most of the connections along the root to leaf path over the multicast tree are done between close peers.

It can be argued that even if the end-to-end latency is not affected a lot by peer failures, fewer peers in the system should lead to a shorter average latency from the root to each destination. However, as we have shown previously in Chapter 4, if we compare the results for sets of 1,000 (Appendix B) and 5,000 peers, MULTI+ works asymptotically better with an increasing number of participants. That means that neither increasing the number of peers increases a lot the average end-to-end latency, nor reducing the number of peers reduces it significantly: the system is not very efficient with few participants. Thus this is something not to blame the recovery procedure for, but MULTI+ itself.

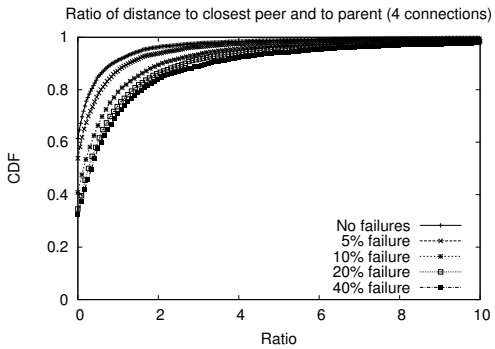
In Figure 5.5 we plot the CCDF (Complementary Cumulative Distribution Function) of the portion of the root-to-leaf latency represented by the *two* largest hop. Notice that our experiments do not consider the first hop coming directly from the source. We see that the more connections we allow, the larger fraction of the total latency is represented by the first two hops. This agrees with the fact that restricting connections increases the number of levels in the tree (Figure 5.2). In the unrestricted tree the two first hops are at least responsible for 70% of the total end-to-end latency for any peer (Figure 5.5(a)). This is not very surprising considering that most peers are in level 2 or 3 of the multicast tree (Figure 5.2(a)). However, the degeneration due to failed peers, increasing the number of levels, does not change the 70% taken by



(a) Unlimited connections.

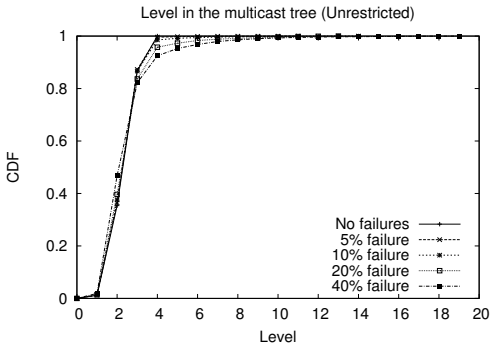


(b) 8 connections.

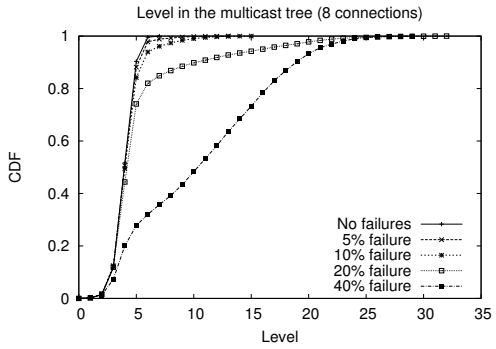


(c) 4 connections.

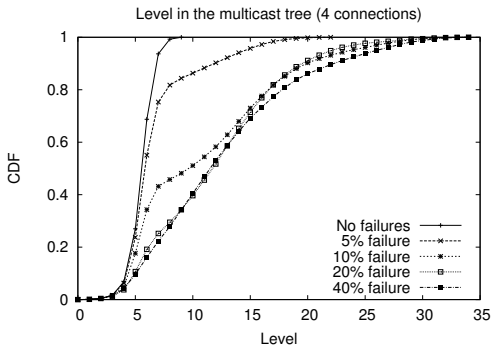
Figure 5.1: Proximity ratio between optimal parent and current one.



(a) Unlimited connections.

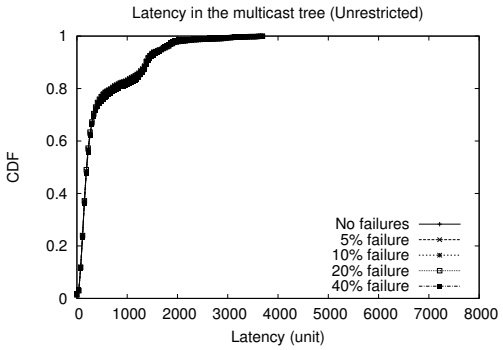


(b) 8 connections.

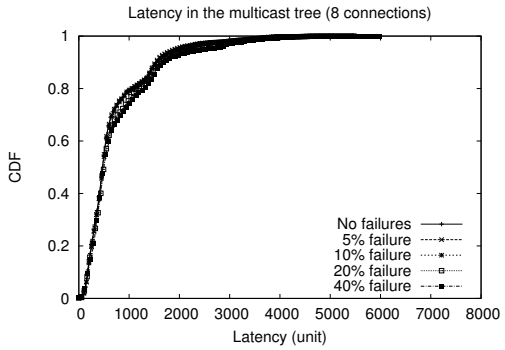


(c) 4 connections.

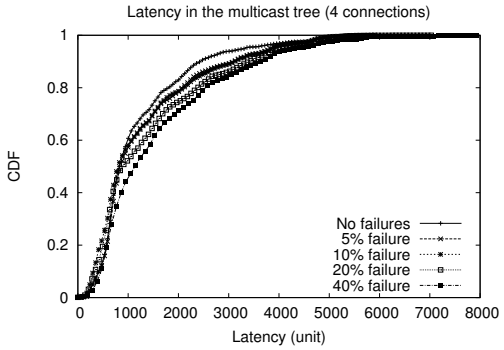
Figure 5.2: Level of peers in the multicast tree.



(a) Unlimited connections.

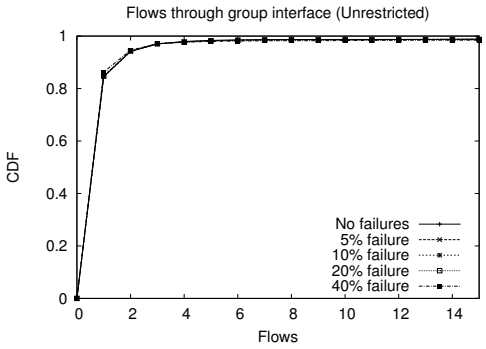


(b) 8 connections.

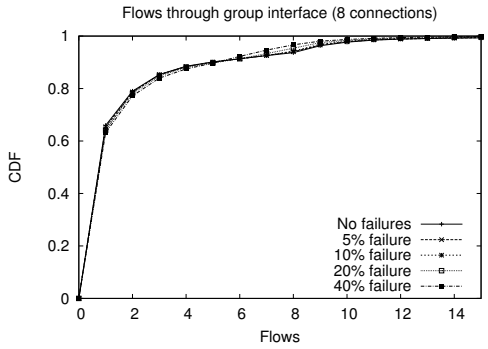


(c) 4 connections.

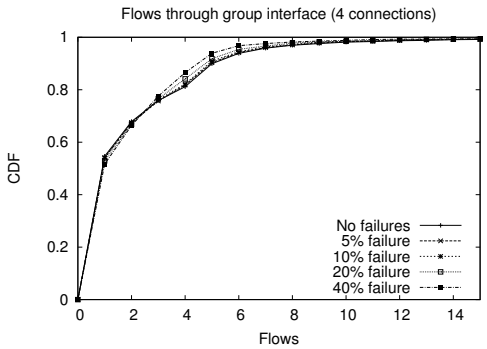
Figure 5.3: Latency from root to leaf (in TC coordinate units) in the multicast tree.



(a) Unlimited connections.



(b) 8 connections.



(c) 4 connections.

Figure 5.4: Number of flows through group interface.

the two largest hops. That means that the increased levels are not adding a lot of latency to the end-to-end total. Thus those new connections are being made to close peers, thanks to the topology-awareness of MULTI+ inherited from TOPLUS. Similar phenomena can be observed when we restrict the number of connections per peer. As an extreme case, see the number of levels in the 4 connection tree when 40% of the peers fail (Figure 5.2(c)), which can get as high as 34. Still (Figure 5.5(c)), the two largest hops for any peer are taking more than 30% of the end-to-end latency. A fifth of the latency introduced by an up to 34 multicast hops is due to two large hops. All the hops in addition to the 8 hops the MULTI+ tree features when no failures occur must thus be between close peers. This effect is the same observed in the topology-aware TOPLUS routing seen in Chapter 3, where a first large hop leads to a destination group, and then each successive hop takes a query closer and closer to the destination peer.

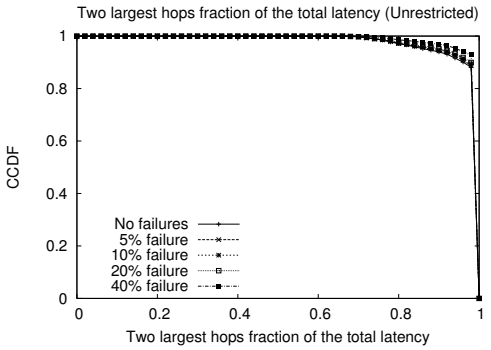
We already knew that MULTI+ is able to make topology-aware tree constructions *inside* TOPLUS groups, that is, network latency towards peers inside a group is assumed to be smaller than latency towards peers outside. However, TOPLUS alone cannot assume anything about peers in different sibling groups. In that case, MULTI+ proceeds connecting to the closest *available* parent using the proximity-aware parent selection. In the next section we study a more straightforward TOPLUS design, and how it affects the performance of MULTI+.

5.5 Hierarchical Internet Partition

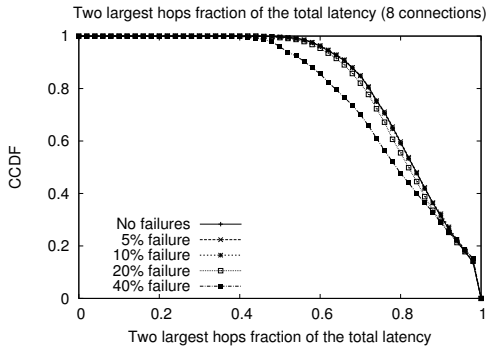
The main problem MULTI+ (or any other TOPLUS-based overlay application) faces is the large amount of state each peer may potentially be forced to maintain. This is what we call the routing table size problem. TOPLUS provides a P2P infrastructure made of peers grouped by IP network prefix obtained from BGP tables, which allows for overlay routing exhibiting a latency similar to that of direct IP routing. TOPLUS routing is closest to IP if we use exclusively the IP network prefixes obtained from BGP tables, forming what we call the *Original* TOPLUS tree (see Chapter 3). However, we get large routing table sizes mainly because in the first (top) tier of the TOPLUS tree we count more than 40,000 groups. Thus each peer should be aware of thousands of other peers. But if we allow that, we could just make each peer know *all* the groups in the TOPLUS tree, and the routing would be done in $O(1)$ hops, thus being identical to that of IP. Of course, in that case the routing tables would grow to impractical sizes. The TOPLUS overlay network protocols are quite light-weight, in the sense that they do not require strict state maintenance. However, a realistic implementation design makes a reduction of routing table sizes mandatory.

5.5.1 A Critique to the TOPLUS Model

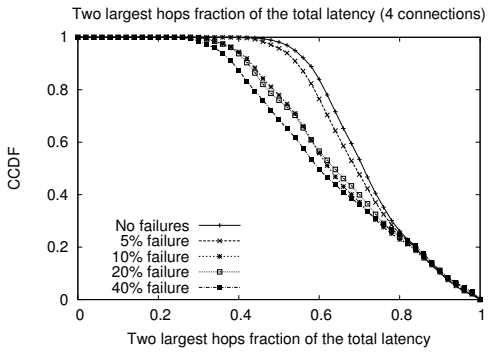
A first solution is to regroup the IP prefixes in tier-1 of the TOPLUS tree using *virtual* groups. We get more reasonable routing table sizes, although that leads to more hops to route a query



(a) Unlimited connections.



(b) 8 connections.



(c) 4 connections.

Figure 5.5: Two largest hops fraction of the end-to-end latency.

to its destination, and consequently, added delay. We call the resulting trees *Modified* TOPLUS trees (as of Section 3.5, 16-bit regrouping, 8-bit regrouping, Original+1, 16-bit+1 and 3-Tier). In principle, we are organizing groups in a convenient way, but we *do not know* if that organization has something to do with the actual topology of the network.

So in the Original tree, if we do not have the information coming from the BGP tables, we do not dare assume anything about the organization of the IP prefixes. In a Modified tree we group prefixes to reduce routing table sizes. But, could we consider, instead of seeing all regrouping as a lack of fidelity to the Internet topology, that in some cases we may be doing the right thing ? The fact is that we may be lacking (we do lack) information about the organization of the whole Internet in every possible corner of it. Indeed, the size and complexity of such a system makes a total knowledge infeasible or impractical (despite the efforts of CAIDA and others). We do not know all BGP tables in the Internet. For the most part, the granularity we get in our TOPLUS trees is coarse, at best.

Thus, we cannot know how much "deviation from the Internet topology" and how much "information missing from our BGP tables" we are adding to the TOPLUS tree when we regroup IP prefixes.

On the other hand, we face situations where we achieve such a precise granularity on the Internet topology that we introduce unnecessary hops in the routing. Tiers and tiers of IP prefixes stack up along network paths that sum up to just a few milliseconds of latency. These very fine-grain groups add nothing, or a very marginal improvement, to the fidelity of the TOPLUS tree to the Internet topology. However, they increase the size of routing tables, and make the interaction between actually close peers cumbersome: the peers in $128.13.125.36/30$ should not need any look-up or whatever other ad-hoc protocol to contact those in $128.13.125.32/30$.

5.5.2 A Practical Solution

The issue just discussed is important, but even more is the fact that in order to build a proper TOPLUS tree the information from BGP tables or other sources (gathering existing network IP prefixes) needs non-trivial amounts of work to be performed by a centralized entity. Moreover, this structure is subject to changes (albeit not frequently), and thus updates are needed. We believe that this approach is feasible, but we propose and test in this chapter a much more straightforward method, the Hierarchical Internet Partition. The Hierarchical Partition tree counts just 3 tiers: the top one with 8-bit prefix groups, the middle one with 16-bit prefix groups and the last with 24-bit prefixes. This establishes a maximum of 1,024 entries in the routing table of a peer (considering also the entries corresponding to the peers in the same inner group). Normally the routing table should be much smaller. This construction closely relates to the 3-Tier tree presented in Chapter 3. The subtle difference between the two is that the 3-Tier tree uses exclusively IP prefixes found in BGP tables, while the Hierarchical Partition includes all possible 8-, 16- and 24-bit prefixes. This allows for more balanced routing tables on all peers, because the overlay network structure perceived by each of them is identical. When routing queries on

such a TOPLUS tree, we obtained the poorest stretch, almost twice the average latency of direct IP routing. However, after the conjectures presented before, we are obliged to at least consider the possibility that some of that increased stretch comes from how different the Hierarchical Partition tree is from the $O(1)$ routing described above, and not exclusively from its lack of topology-awareness. That is, the stretch can increase due to the fact that we have to make more hops to reach a destination when routing (because groups are organized in a deeper hierarchy), and not only from the lack of topology-awareness of those hops.

We perform a series of experiments in order to measure the relative locality of peers inside each group of the Hierarchical Partition. We call locality "relative" because we compare the average latency among peers inside a group to the average latency of those peers to all other peers outside the group. We do this for each group at each tier, thus for all 8-bit prefix (tier-1), 16-bit prefix (tier-2) and 24-bit prefix (tier-3). We simulate a deployment of 5,000 peers from the TC address space, using the TC coordinates to measure latencies among them.

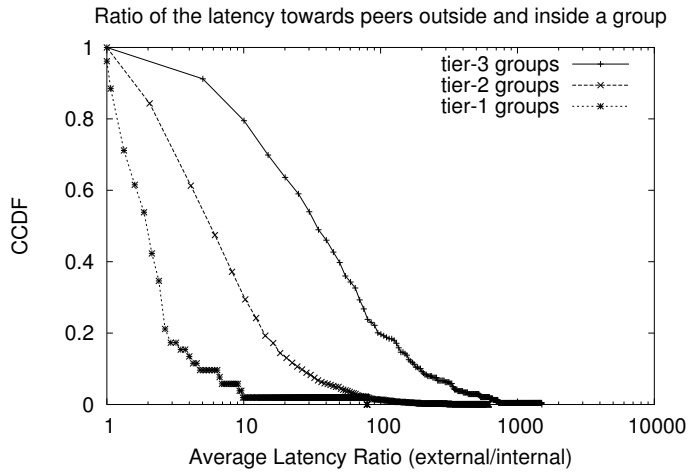


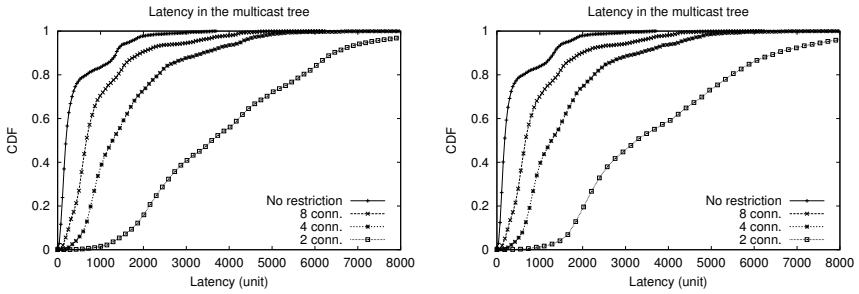
Figure 5.6: Ratio of the average latency of peers in a group towards peers in other groups to the average latency among peers in that group.

In Figure 5.6 we observe that the deeper in the hierarchy, and thus the longer the prefix, the stronger the difference between latency to peers in the same group and to those outside the group. For example, we see in Figure 5.6 that 80% of the groups in tier-3 present latencies between the peers inside them at least 10 times smaller than the latencies towards the peers in other groups. Thus this simple hierarchical partitioning of the IP address space seems to inherit the properties of the IP prefix hierarchy obtained from BGP tables. In some cases, the 256 potential peers per inner group may be a lot to manage, specially in low-bandwidth clients. However, the good locality inside a 24-bit prefix group allows for an arbitrary partition of the group in, for instance, 28-bit prefix groups adding a final fourth tier to the tree.

We present now results for the tests studying MULTI+ on the Hierarchical Partition TOPLUS tree. As we will see, these results are very similar to those obtained using the Modified tree, although they offer a slightly worse performance, even when proximity-aware algorithms are

used.

As one can see in Figure 5.7 (we compare the root-to-leaf delay as the most significant figure of merit, others present similar behavior), the average loss in performance is not significant, while the simplicity gained by avoiding the periodic retrieval and distribution of the IP prefix tree is an important improvement. It is worth noting that in Figure 5.7 we use the FIFO parent selection, so no active measurement is done at all.



(a) Using the Modified TOPLUS tree.

(b) Using the Hierarchical Partition TOPLUS tree.

Figure 5.7: Latency from root to leaf (in TC coordinate units) in the multicast tree.

To us, these results indicate that the Hierarchical Partition tree is a feasible alternative to the other configurations previously studied. The Hierarchical Partition tree solves (or at least severely mitigates) the routing table size problem and offers good performance on MULTI+, at least when compared to the others TOPLUS trees.

5.6 Planet-Lab Experiments

In this final section we test the concepts developed so far on Planet-Lab. We do not deploy a real client, but we study the quality of the multicast trees created by MULTI+ on a set of Planet-Lab hosts.

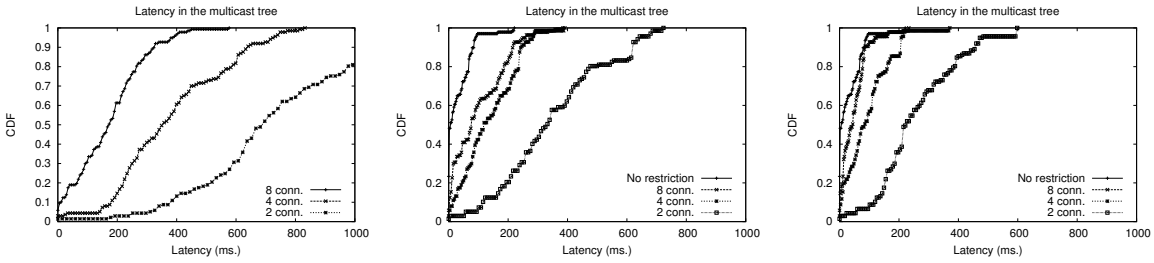
The latencies calculated from the TC coordinate space present a deviation from the real figures. We do not feel that the error present in the process of abstraction from the real latencies in the Internet to the position of hosts in the coordinate space affects seriously our results. This deviation between coordinates and real latencies has been measured and bounded. Still, Planet-Lab offers a few hundred hosts on the Internet and the possibility of getting real latencies from a real environment to test our algorithms.

We obtain a full list of IP addresses of hosts in Planet-Lab, at the time of these tests, 328. This is a considerable set, but in previous tests we already saw a significant improvement in the multicast tree properties when increasing the host population from 1,000 to 5,000 peers. We

measure the latency from each host to the others using `ping`, four times a day, during the two weeks from April the 14^{th} to the 30^{th}, 2004. We establish the latency between two hosts as the lowest one measured during those two weeks.

We finally obtain a full mesh of 137 hosts. This is not a massive multicast client population, but the value of this set is that all parameters are directly obtained from a real deployment of hosts: IP address distribution and latencies between hosts. We insist so much on the need for a full latency mesh because we do not know *a priori* how peers may connect to each other. When the tree is built, if the latency between two hosts is not known we cannot decide on the quality of the decision taken. We could work with a larger setup of peers, constraining the multicast tree to follow only the links we have been able to measure. But we do not want our results to be affected by such an arbitrary constraint.

With the data gathered, we compare the results using the Modified TOPLUS tree presented before and the Hierarchical Partition tree. We obtain satisfying results testing MULTI+ on a real set of hosts in Planet-Lab. The Hierarchical Partition tree seems to be a good approximation of the network topology. One can compare the improvement in end-to-end delay in the overlay network from Figure 5.8(a), where each peer connects to a randomly chosen parent, to Figure 5.8(b) where no measurement at all is performed, just the MULTI+ algorithm is used on top of TOPLUS. Still, figures are not as impressive as those using 5,000 peers (e.g., Figure 5.3). Indeed, the Hierarchical Partition, or any other TOPLUS layout for that matter, is a large infrastructure covering the whole IP address space. The aggregated properties of MULTI+ improve as the peer population increases, allowing multicast streams to really benefit from the topology-awareness. The more peers present inside a group, the higher benefit a single flow provides into that group.



(a) Random parent selection.

(b) FIFO parent selection.

(c) Proximity-aware parent selection.

Figure 5.8: Latency from root to leaf (in milliseconds) in the multicast tree over a Hierarchical Partition of Planet-Lab machines.

Small sparse peer populations can not obtain much advantage of our approach, although, as this experiment shows, one or two hundred peers is a large enough critic mass.

Our tests show that the group interfaces (IP network prefixes) in the TOPLUS tree obtained from BGP tables are in average 10% more loaded (each interface sees more flows going through)

when using the Hierarchical Partition tree, because of its lack of detail on the real network organization. This may rise an issue for very large deployments counting millions of peers, where a precise knowledge of the network topology is necessary to avoid inefficient bandwidth utilization. For relatively small deployments, we expect to avoid bottlenecks due to the inaccuracy of the fixed-size prefix partitioning thanks to the random character of the parent selection among the available peers inside each fixed prefix. The multicast flows from and towards a given prefix of the Hierarchical Partition should be distributed over all the existing peers (and thus over the "real" prefixes) inside.

5.7 Conclusion

MULTI+ relies on TOPLUS in order to find a proper parent for a peer in the multicast tree. Instead of using BGP tables information to obtain a fine-grained TOPLUS tree, we have presented an approximative TOPLUS tree that severely improves the feasibility of an hypothetical MULTI+ implementation. MULTI+ maintains the advantage of being able to create topology-aware multicast trees without introducing extra traffic for active measurement even when network prefix information may not be available. Although the out-of-band information regarding the TOPLUS routing tables can be calculated offline and downloaded just like many P2P systems today require to download a list of peers for the join process, the Hierarchical Internet Partition seems to be good compromise, providing sufficient accuracy while simplifying a lot the overlay construction. MULTI+ remains robust under massive peer failure, and it is able to maintain low end-to-end delay even in case of a limited number of connections per peer. These interesting features are achieved without the need for probing or widely measuring, just using a small set of algorithms and the properties of a TOPLUS overlay network.

Chapter 6

Data Indexing in DHT-based Peer-to-Peer Systems

We should have a great fewer disputes in the world if words were taken for what they are, the signs of our ideas only, and not for things themselves.

John Locke. Essay, "Of Words"

Peer-to-peer systems with Distributed Hash Table look-up make specific data discovery simple when the complete identifiers of data —or keys— are known in advance. In practice, however, users looking up resources stored in peer-to-peer systems often have only partial information for identifying these resources. In this chapter, we describe techniques for indexing data stored in DHT-based P2P systems, and discovering the resources that match a given user query. Our system creates multiple indices, organized hierarchically, which permit users to locate data even using scarce information, although at the price of a higher look-up cost. The data itself is stored on only one (or few) of the peers. Experimental evaluation demonstrates the effectiveness of our indexing techniques on a distributed P2P bibliographic database with realistic user query workloads.

6.1 Introduction

Peer-to-peer (P2P) systems make it possible to harness the computing power and resources of large populations of networked computers in a cost-effective manner. In this chapter, we focus on P2P systems where data items are spread across many different peer computers and the location of each item is determined in a decentralized manner using a Distributed Hash Table (DHT) look-up service.

A major limitation of DHT look-up services is that they only support exact-match look-ups: one needs to know the exact key (identifier) of a data item to locate the peer responsible for that item. In practice, however, P2P users often have only partial information for identifying these items and tend to submit broad queries [27] (e.g., all the articles written by "John Smith").

In this chapter, we propose to augment DHT-based P2P systems with mechanisms for locating data using incomplete information. Note that we do not aim at answering complex database-like queries, but rather at providing practical techniques for searching data with a DHT look-up. Our mechanisms rely on indices, stored and distributed across the peers of the network, that maintain useful information about user queries. Given a broad query, a user can obtain additional information about the data items that match his original query; the DHT look-up service can be recursively queried until the user finds the desired data items. Indices can be organized hierarchically to reduce space and bandwidth requirements, and to facilitate interactive searches. They integrate an adaptive distributed cache to speed up accesses to popular content.

Our indexing techniques can be layered on top of an arbitrary DHT look-up service, and thus benefit from any advanced features implemented in the DHT (e.g., replication, load-balancing). We have conducted a comprehensive evaluation that demonstrates their effectiveness in realistic settings. In particular, we have observed that our techniques are scalable, adapt well to user search patterns, and have reasonably-small space requirements. Look-up times depend on the "precision" of the initial query: broad queries incur higher look-up times than specific queries.

The organization of this chapter is as follows: In Section 6.2, we discuss related work, and we introduce the system model in Section 6.3. We describe our distributed indexing techniques in Section 6.4 and we evaluate them in Section 6.5. Finally, Section 6.6 concludes the chapter.

6.2 Related Work

Of the various related project, INS/Twine [128] is most similar to our work. INS/Twine is an architecture for intentional (i.e., based on what we are looking for, not where it is located [129]) resource discovery, which allows to easily locate services and devices in large scale environments, using intentional descriptions. INS/Twine works on top of a DHT, such as Chord [33], by setting up a number of resolvers, which collaborate to distribute resource information and to resolve simple queries.

Given a semi-structured resource description, INS/Twine extracts prefix subsequences of attributes and values, called "strands". INS/Twine then computes the hash values for each of these strands, which constitutes numeric keys used to map resources to resolvers. The resource and device information are stored redundantly on *all* resolver peers that correspond to the numeric keys. When looking up some resource, INS/Twine sends the query to the resolver peer identified by one of the longest strands; the query is further processed by the resolver, which returns the matching resource descriptions.

Unlike Twine, we do not replicate data at multiple locations; we rather provide a key-to-key service, or more precisely a query-to-query service. We do not introduce dedicated resolvers in our architecture; we only require the underlying distributed data storage system to allow for the registration of multiple entries using the same key. As we allow index keys to be tree-structured and not necessarily prefix sub-keys, data can be looked up using more expressive and selective queries. For improved scalability, index entries are further organized hierarchically.

In [130], Harren *et al.*discuss techniques for performing complex queries in DHT-based P2P networks, using traditional relational database operators (selection, projection, join, grouping and aggregation, and sorting) and elaborate text retrieval techniques (like splitting a query string and using each piece to create a key matching the query).

This technical report [131] from Gupta *et al.*develop a P2P data sharing architecture for computing approximate answers for complex queries by finding data ranges that are similar to the user query. Relevant data is located using "locality sensitive hashing" techniques. In [132], the same authors extend the CAN [40] system to support the basic range operation on data shared in the form of database relations. The construction of a "similarity function" is left open in this approach [133], which organizes the fields describing an object in a hierarchy. A set of values describing an object is located the lower in the hierarchy the more concise the characterization of the object.

More recently, the authors of [34] combine the flexibility of searches in unstructured P2P networks (like Gnutella [18]) with peer grouping to form associative overlays. Peers are organized in groups with common interests. Groups are located by routing queries, and searches are performed within groups using more flexible algorithms. In [134], the authors question the feasibility of an Internet wide search engine based on P2P, but their conclusion do not apply to our techniques, which are more aimed at smaller scale systems with well-specified content.

pSearch [135] uses vector representation of documents to gather related items in the same search space. Their approach deepens much more in semantic search than our exact match method, providing a possible extension for our work. Reynolds and Vahdat [136] use a method resembling ours, but they index all documents matching a given keyword on the corresponding peer in the DHT. Our flat indexing scheme (to be shown later in this chapter) is similar to their approach. Another proposal is CANDy [137], which uses logical and set operators to build more complex queries.

Other efforts of interest in P2P data location can be found in [138].

6.3 System Model and Definitions

The distributed indices are layered on top of a DHT-based P2P network —or substrate—and uses various other technologies to describe content and represent user queries. We now describe

the specific requirements of our indexing techniques.

6.3.1 P2P Storage System

A DHT look-up maps keys to peers in a peer-to-peer infrastructure. Any peer can use the DHT look-up service to determine the current live peer that is responsible for a given key. In this chapter, we assume an underlying DHT-based P2P data storage system, in which each data item is mapped to one or several peers. Example of such systems are Chord/DHash/CFS [60] and Pastry/PAST [59].

Throughout the chapter, we will use the example of a bibliographic database system that stores scientific articles. Files are identified by *descriptors*, which are textual, human-readable descriptions of the file's content (as in Figure 6.1). Let $h(descriptor)$ be a hash function that maps identifiers to a large set of numeric keys. The peer responsible for storing a file f is determined by transforming the file's descriptor d into a numeric key $k = h(d)$. This numeric key is used by the DHT look-up service to determine the peer responsible for f. In order to find f, a peer p has to know the numeric key or the complete descriptor.

```
<article>
  <author>
    <first>John</first>
    <last>Smith</last>
  </author>
  <title>TCP</title>
  <conf>SIGCOMM</conf>
  <year>1989</year>
  <size>315635</size>
</article>
```

d_1

```
<article>
  <author>
    <first>John</first>
    <last>Smith</last>
  </author>
  <title>IPv6</title>
  <conf>INFOCOM</conf>
  <year>1996</year>
  <size>312352</size>
</article>
```

d_2

```
<article>
  <author>
    <first>Alan</first>
    <last>Doe</last>
  </author>
  <title>Wavelets</title>
  <conf>INFOCOM</conf>
  <year>1996</year>
  <size>259827</size>
</article>
```

d_3

Figure 6.1: Sample File Descriptors.

6.3.2 Data Descriptors and Queries

We assume that descriptors are semi-structured XML data [139, 140], as used by many publicly-accessible databases (e.g., DBLP [141]). Examples of descriptors for bibliographic data are given in Figure 6.1. These descriptors have fields useful for searching files (e.g., author, title), as well as fields useful for an administrator of the database (e.g., size).

To search for data stored in the Peer-to-peer network, we need to specify broad queries that can match multiple file descriptors. For this purpose, we use a subset of the XPath XML addressing language [142], which offers a good compromise between expressiveness and simplicity. XPath treats XML documents as a tree of nodes and offers an expressive way to specify and select parts of this tree. An XPath expression contains one or more *location steps*, separated

by slashes (/). In its more basic form, a location step designates an element name followed by zero or more predicates specified between brackets. Predicates are generally specified as constraints on the presence of structural elements, or on the values of XML documents using basic comparison operators. XPath also allows the use of wildcard (*) and ancestor/descendant (//) operators, which respectively match exactly one and an arbitrarily long sequence of element names. An XML document (i.e., a file descriptor) *matches* an XPath expression when the evaluation of the expression on the document yields a non-null object.

```
q1 = /article[author[first/John][last/Smith]] ···
     [title/TCP][conf/SIGCOMM][year/1989][size/315635]
q2 = /article[author[first/John][last/Smith]][conf/INFOCOM]
q3 = /article/author[first/John][last/Smith]
q4 = /article/title/TCP
q5 = /article/conf/INFOCOM
q6 = /article/author/last/Smith
```

Figure 6.2: Sample File Queries.

For a given descriptor d, we can easily construct an XPath expression (or query) q that tests the presence of all the elements and values in d.[1] We call this expression the *most specific query* for d or, by extension, the *most specific descriptor*. Conversely, given q, one can easily construct d, compute $k = h(d)$, and find the file. For instance, query q_1 in Figure 6.2 is the most specific query for descriptor d_1 in Figure 6.1.

Given two queries q and q′, we say that q′ *covers* q (or q is covered by q′), denoted by $q' \sqsupseteq q$, if any descriptor d that matches q also matches q′. Abusing the notation, we often use d instead of q when q is the most specific query for d and we consider them as equivalent ($q \equiv d$); in particular, we say that q′ covers d when $q' \sqsupseteq q$ and q is the most specific query for d.

It should be noted that the covering relation introduces a partial ordering on the queries. The partial ordering graph for queries in Figure 6.2 is shown in Figure 6.3, where $q_i \rightarrow q_j$ is read $q_i \sqsupseteq q_j$ (more specific queries are represented above less specific queries).

6.4 Indexing Algorithm

When the most specific query for the descriptor d of a file f is known, finding the location of f is straightforward using the key-to-peer (and hence key-to-data) underlying DHT look-up

[1] In fact, we can create several equivalent XPath expressions for the same query. We assume that equivalent expressions are transformed into a unique normalized format.

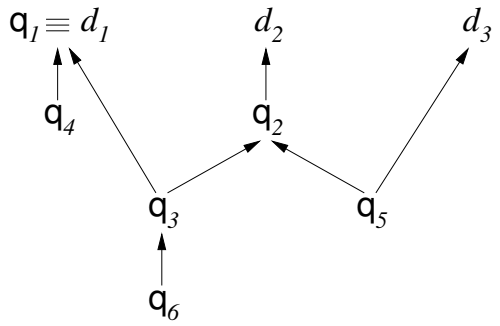


Figure 6.3: Partial ordering tree for the queries of Figure 6.2 (self-covering and transitive relations are omitted).

service. The goal of our architecture is to also offer access to f using less specific queries that cover d.

The principle underlying our technique is to generate multiple keys for a given descriptor, and to store these keys in indices maintained by the DHT in the P2P system. Indices do not contain key-to-data mappings; instead, they provide a key-to-key service, or more precisely a query-to-query service. For a given query q, the index service returns a (possibly empty) list of more specific queries, covered by q. If q is the most specific query of a file, then the P2P storage system returns the file (or indicates the peer responsible for that file). By iteratively querying the index service, a user can traverse upward the partial order graph of the queries (see Figure 6.3) and discover all the indexed files that match his broad query.

In order to manage indices, the underlying P2P storage system must be slightly extended. Each peer should maintain an index, which essentially consists of query-to-query mappings. The "$insert(\mathsf{q},\mathsf{q}_i)$" function, with $\mathsf{q} \sqsupseteq \mathsf{q}_i$, adds a mapping $(\mathsf{q};\mathsf{q}_i)$ to the index of the peer responsible for key q. The "$look-up(\mathsf{q})$" function, with q not being the most specific query of a file, returns a list of all the queries q_i such that there is a mapping $(\mathsf{q};\mathsf{q}_i)$ in the index of the peer responsible for key q.

Roughly speaking, we store files and construct indices as follows: Given a file f and its descriptor d, with a corresponding most specific query q, we first store f at the peer responsible for the key $k = h(\mathsf{q})$. We generate a set of queries $\mathsf{q} = \{\mathsf{q}_1, \mathsf{q}_2, \ldots, \mathsf{q}_l\}$ likely to be asked by users (to be discussed shortly), and such that each $\mathsf{q}_i \sqsupseteq \mathsf{q}$. We then compute the numeric key $k_i = h(\mathsf{q}_i)$ for each of the queries, and we store a mapping $(\mathsf{q}_i;\mathsf{q})$ in the index of the peer responsible for each k_i in the P2P network. We iterate the process shown for q to every q_i, and we continue recursively until all the desired index entries have been created.

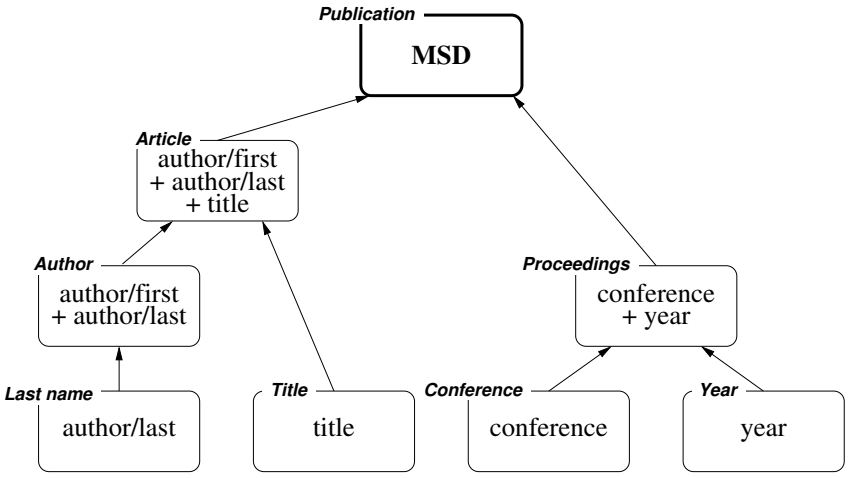


Figure 6.4: Sample indexing scheme for a bibliographic database.

6.4.1 Indexing Example

To best illustrate the principle of our indexing techniques, consider a P2P bibliographic database that stores the three files associated to the descriptors of Figure 6.1. We want to be able to look up publications using various combinations of the author's name, the title, the conference, and the year of publication. A possible hierarchical indexing scheme is shown in Figure 6.4. Each box corresponds to a distributed index, and indexing keys are indicated inside the boxes. The index at the origin of an arrow stores mapping between its indexing key and the indexing key of the target. For instance, the *Last name* index stores the full names of all authors that have a given last name; the *Author* index maintains information about all articles published by a given author; the *Article* index stores the descriptors (MSDs) of all publications with a matching title and author name. After applying this indexing scheme to the three files of the bibliographic database, we obtain the distributed indices shown in Figure 6.5. The top-level *Publication* index corresponds to the raw entries stored in the underlying P2P storage system: complete key provide direct access to the associated files. The other indices hold query-to-query mappings that enable the user to iteratively search the database and locate the desired files. Each entry of an index is potentially stored on a different peer in the P2P network, as illustrated for the *Proceedings* index. One can observe that some index entries associate a query to multiple queries (e.g., in the *Author* index).

Figure 6.6 details the individual query mappings stored in the indices of Figure 6.5. Each arrow corresponds to a query-to-query mapping, e.g., $(q_6; q_3)$. The files corresponding to descriptors d_1, d_2, and d_3 can be located by following any valid path in the partial order tree. For instance, given q_6, a user will first obtain q_3; the user will query the system again using q_3 and obtain two new queries that link to d_1 and d_2; the user can finally retrieve the two files matching its query using d_1 and d_2.

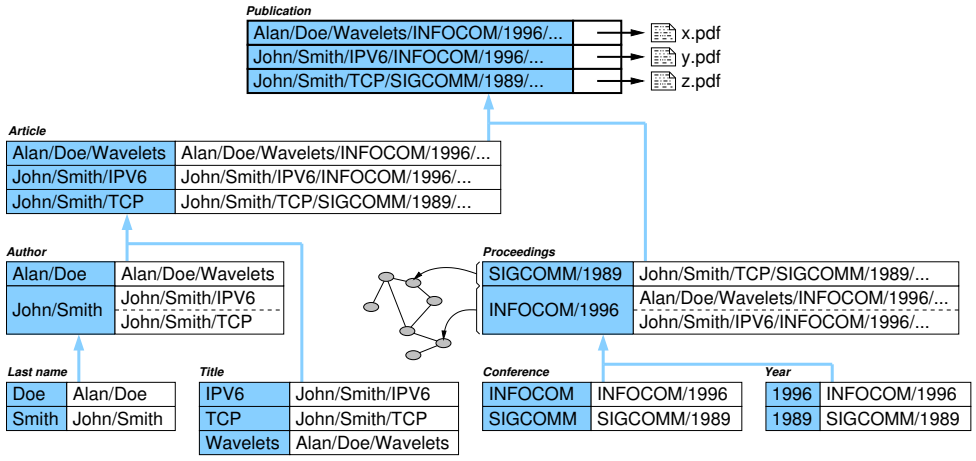


Figure 6.5: Sample distributed indices for the three documents of Figure 6.1 and the indexing scheme of Figure 6.4 (query syntax has been simplified).

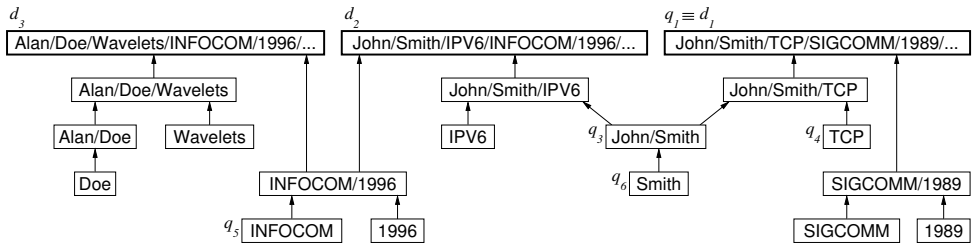


Figure 6.6: Query mappings for the indices of Figure 6.5 (identifiers correspond to Figures 6.1 and 6.2; query syntax has been simplified).

6.4.2 Look-Ups

We can now describe the look-up process more formally. When looking up a file f using a query q_0, a user first contacts the peer p responsible for $h(q_0)$. That peer may return f if q_0 is the most specific query for f, or a list of queries $\{q_1, q_2, \ldots, q_n\}$ such that the mappings $(q_0; q_i)$, with $q_0 \sqsupseteq q_i$, are stored at p. The user can then choose one or several of the q_i and repeat this process recursively until the desired files have been found. The user effectively follows an "index path" that leads from q_0 to f ("guided tour").

Look-ups may require several iterations when the most specific query for a given file is not known. Higher index hierarchy usually necessitate more iterations to locate a file, but are also generally more space-efficient, as each index factorizes in a compact manner the queries of its child indices. In particular, the size of the lists (result sets) returned by the index service may be prohibitively long when using a flat indexing scheme (consider, for example, the list of all articles written by the persons whose last name is "Smith"). There is therefore a trade-off between space requirements, size of result sets, and look-up time, as we shall see in the experimental evaluation.

The look-up process can be interactive, i.e., the user directs the search and restricts its query at each step, or automated, i.e., the system recursively explores the indices and return all the file descriptors that match the original query.

When a user wants to look up a file f using a query q_0, it may happen that q_0 is not present in any index, although f does exist in the peer-to-peer system and q_0 is a valid query for f. Ii is still possible to locate f, by (automatically) looking for a query q_i such that $q_i \sqsupseteq q_0$ and q_i is on some index path that leads to f.

For instance, given the distributed indices of Figures 6.5 and 6.6, it appears that query q_2 in Figure 6.2 is not present in any index (q_2 =`/article[author[first/John][last/Smith]][conf/INFOCOM]`). We can however find q_3, such that $q_3 \sqsupseteq q_2$ and there exists an index path from q_3 to d_1. Therefore, the file associated to d_1 can be located using this generalization/specialization approach, although at the price of a higher look-up cost. We believe that it is natural for look-ups performed with less information to require more effort.

6.4.3 Building and Maintaining Indices

When a file is inserted in the system for the first time, it has to be indexed. The choice of the queries under which a file is indexed is arbitrary, as long as the covering relation holds. As files are discovered using the index entries, a file is more likely to be located rapidly if it is indexed "enough" times, under "likely" names. The quantity and likelihood of index queries are hard to quantify and are often application-dependent. For instance, in a bibliographic database, indexing a file by its size is useless for users, as they are unlikely to know the size beforehand.

However, indexing the files under the author, title, and/or conference are appropriate choices.

Note that the length of the index paths that lead to a given file is arbitrary, although it directly affects the look-up time. Less popular content may be indexed using a deeper index hierarchy, to reduce space and bandwidth requirements, and to facilitate interactive searches. In contrast, a very popular file can be linked to deep in the hierarchy to short-circuit some indices and speed up look-ups. For instance, given the distributed indices of Figures 6.5 and 6.6, one can add the $(q_6; d_1)$ index entry to speed up searches for the popular file described by d_1 (e.g., the author's most popular publication).

Note also that more generic queries can be obtained from more specific queries by removing only portions of element names (i.e., using substring matching). For instance, one can create an index with all the files of an author that start with the letter "A", the letter "B", etc. One can also envision to use techniques similar to those discussed in [130] for substring matching. In general, determining good decompositions for indexing each given descriptor type (e.g., articles, music files, movies, books, etc). requires human input.

In a system model where files are injected in the system, but are never deleted (write-once semantics), index entries never need to be updated. In a read/write system, when a file is deleted we have to find all the indices that refer to the descriptor of that file and delete the associated mappings. Locating the index entries can be achieved straightforwardly by using the same process used to generate them in the first place when the file was injected in the system. When deleting the last mapping for a given key, we can recursively delete the references to that key to clean up the indices.

Index entries can also be created dynamically to adapt to the query patterns of the users. For instance, a user who tries to locate a file f using a non-indexed query q_0, and eventually finds it using the query generalization/specialization approach discussed above, can add an index entry to facilitate subsequent look-ups from other users.

More interestingly, one can easily build an adaptive cache in the P2P system to speed up accesses to popular files. Assume that each peer allocates a limited number of index entries for caching purposes. After a successful look-up, a peer can create "shortcuts" entries (i.e., direct mappings between generic queries and the descriptor of the target file) in the caches of the indices traversed during the look-up process. Another user looking for the same file via the same path will be able to "jump" directly to the file by following the shortcuts stored in the caches. With a least-recently used (LRU) cache replacement policy (i.e., the least used entries in the cache are replaced by the new ones once there is no cache space left), we can guarantee that the most popular files are well represented in the caches and are accessible in few hops. The caching mechanism therefore adapts automatically to the query patterns and file popularity.

6.4.4 Advantages of Indexing

We outline below some interesting properties of our indexing techniques, which we will further study in the experimental evaluation:

- *Space efficient:* First, as indices contain key-to-key mappings, the data items do not have to be stored on multiple peers but the queries (or variants). Second, although data items may be reached through multiple index paths, the space requirements remain reasonably small because coarse-level indices are shared by many data items (e.g., given the mappings of Figure 6.6, $(q_6;q_3)$ is on index paths to both d_1 and d_2). Finally, the hierarchical index organization significantly reduces the size of results sets, and consequently the bandwidth requirements.
- *Scalability:* As data items may be accessed through distinct paths and are referred to in distinct indices, the load is expected to be spread across multiple indices, and thus multiple peers (in contrast to a centralized index). In addition, since indices are stored as regular data items, they can benefit from the mechanisms implemented by the DHT-based P2P network for increasing availability and scalability, such as data replication or caching. The cache entries suggested above need not be treated as data stored in the P2P system, because they are not essential for the indexing to work; cache is just a performance booster, and it can be regenerated if lost due to the failure of a peer.
- *Loose coupling between data and indices:* When the data items change, only the peers responsible for the complete key of the data need to be updated. Indices do not need to be updated. This is a consequence of the key-to-key mapping technique.
- *Versatility:* It is possible not to index some data, and enforce access using the complete key. Conversely, some popular data may be aggressively indexed to speed up accesses.
- *Adaptability:* The system can create index entries on-demand to match user querying habits or for caching purposes.
- *Resilient to arbitrary linking:* When inserting a file in the system, it can only be indexed at locations that correspond to keys covering the file's key. Arbitrary links (or aliases) to a file cannot be inserted in the system, provided the correctness of covering relations between indices is enforced by peers. This makes it harder for a user to inject a file with malicious or offensive content and masquerade it as another existing file.

6.5 Evaluation of Data Indexing in P2P Systems

The indexing mechanism lies in the protocol stack on top of a storage application on a P2P network. Thus, one aspect of indexing performance deals with the optimization of resources offered by those lower layers (see Figure 1.2 in Chapter 1). As our indexing techniques do not

depend on a specific look-up and storage layer, we do not explicitly study the performance of the P2P network.

The index protocol layer also interacts with the end user, who expects to find data of interest using partial information. From the user point of view, it is clearly desirable to find the desired data in a minimal number of interactions with the system, and the information returned by the system should be as concise and relevant as possible. From the system point of view, the search process should be simple, the amount of network traffic should be minimized, and the storage space dedicated to the indexing metadata should remain within reasonable limits. These various criteria are studied in the rest of this section.

6.5.1 Distributed Bibliographic Database

To study the behavior of a P2P indexed network, we model a bibliographic database distributed among interconnected hosts. The bibliographic database contains articles published in journals and conference proceedings. The underlying P2P look-up and storage system, and the physical characteristics of the network, are not important: we simply assume that the underlying DHT look-up service is able to find a peer p responsible for a given key k, where p stores (or knows the location of) the data associated with key k.

In order to build the bibliographic database, we used the publicly-available DBLP [141] archive, which consists of an XML-formatted list of publications. The DBLP archive, as of January 21^{st}, 2003, contains more than $346,000$ entries: articles, theses, proceedings, etc. For our experiments we used the $115,879$ article entries in the archive to build the descriptors (MSDs) of the corresponding articles. The archive being in XML format, entries are pretty similar to those in Figure 6.7 (actual field names differ).

```
<article key="tr/dec/SRC1997-025">
<editor>John</editor>
<title>SQL</title>
<journal>Digital</journal>
<volume>SRC1997-025</volume>
<year>1997</year>
<cdrom>src1997-025.pdf</cdrom>
</article>
```

Figure 6.7: An article XML entry from the DBLP archive.

6.5.2 Building Indices

From the MSD, which actually links to the real data, we build chains of queries, i.e., sequences of queries where each query covers ($\sqsupseteq$) the next one. The last member of each chain is obviously the MSD, which is covered by all the previous ones. Each chain corresponds to a path from a leaf to a root in Figure 6.3.

To create indices that correspond to queries that users are likely to ask, we have analyzed the query logs of two other bibliographic database sites: BibFinder [143] and NetBib [144] (the DBLP service does not store or share query logs). NetBib offers an interface where a user can search for papers using fields like: words in title or abstract, exact title, author, publication date (year intervals), and citation key. BibFinder displays a similar interface: a user can issue queries with the author name, title, conference or journal, and year of publication (exact, or published before/after a given year).

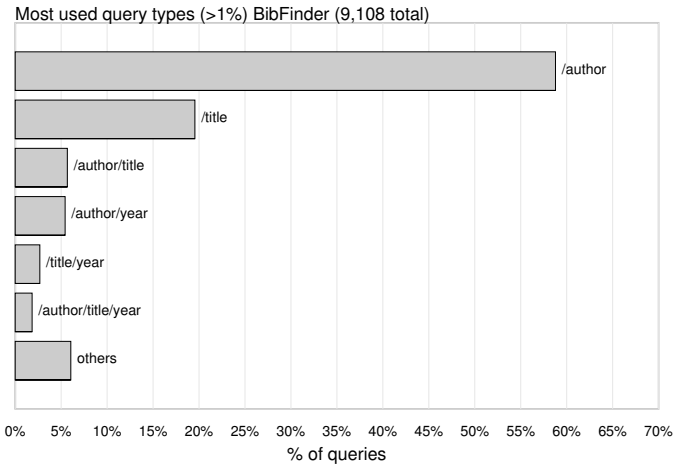


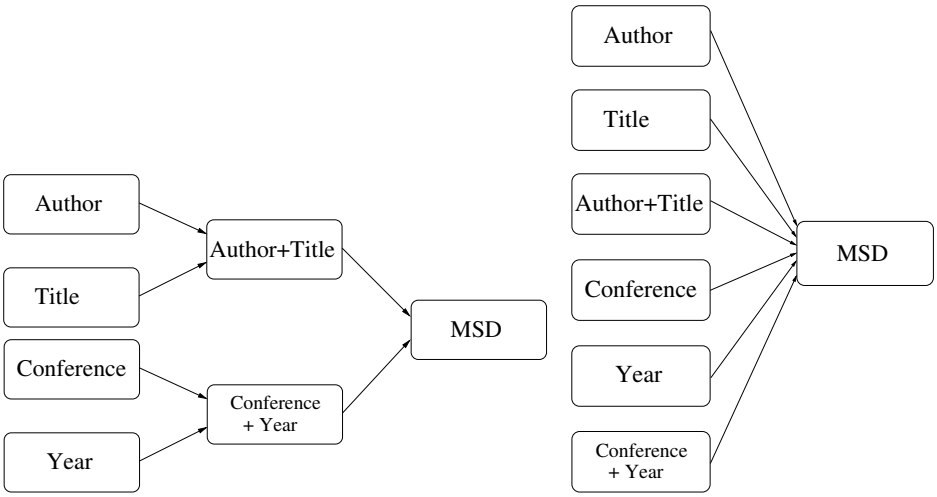
Figure 6.8: Distribution of the types of queries extracted from BibFinder's log.

The log from BibFinder's site contains 9,108 queries. As shown in Figure 6.8, most of them use only the "author" field (57%), followed by those using only the "title" field (20%), and those with a given publication date.

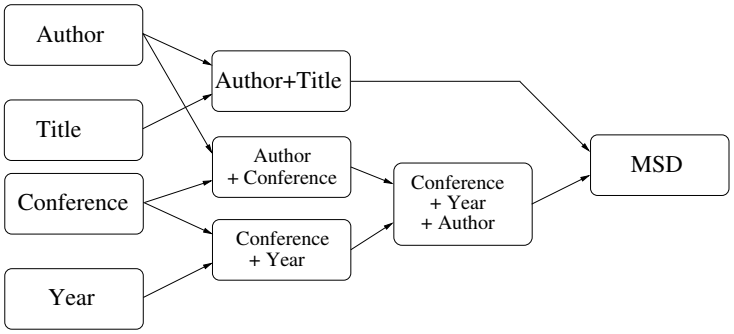
The NetBib trace represents 5,924 different queries. Similarly to the BibFinder traces, there are three main kinds of queries (summing up to more than 95% of the total): queries for a given author, topic (title), and date of publication.

Based on this information, we have simulated and compared the following three indexing schemes, shown in Figure 6.9.

- *Simple:* A query for an author or a title returns a set of author and title pairs. Each of



(a) Simple. (b) Flat.



(c) Complex.

Figure 6.9: Indexing schemes.

them points to a MSD. A query for a conference or a year returns a set of conference-year pairs, in turn pointing to MSDs.

- *Flat:* All possible queries in the *simple* scheme point directly to the MSD, so that the index query length is always 2.
- *Complex:* Some queries in the *simple* scheme are split into more specific queries, in order to avoid long result lists. For instance, a query specifying an author and a conference returns a list of queries that further indicate all the publication years for the given author and conference. Although we do not expect index hierarchies to be very deep in practice, this scheme allows us to observe the effect of hierarchy depth.

The *simple* indexing scheme is the most space-efficient of the three, requiring 152 MB of extra storage in the system for the full DBLP article collection. The *complex* scheme requires 25% more space, and the *flat* scheme a 37% increase, being the most space-consuming.

By comparison, we have evaluated the cost of storing the actual article files in the P2P infrastructure. Based on an average file size of 250 KB (estimated from the size of the articles in our own private digital library, counting more than 7,000 files in Postscript and Adobe PDF format), 29.1 GB are required to store all the articles in the DBLP archive. In the worst case, the indices require an additional storage capacity of 0.5%. For a P2P system storing the full article collection, the additional space required for index storage is clearly negligible.

6.5.3 User Model

The mean number of interactions needed to reach an article gives a measure of the efficiency of a given indexing scheme. This can be computed as the average length of all possible query chains a user can follow to find an article. This is, however, only valid in an scenario where all kinds of queries are used with equal probability, and all articles are equally requested; such assumptions are not realistic.

To model realistic user behavior, we built a query generator that reflects which information is used to build queries (query structure), and which data is requested (data popularity).

Realistic query structure model

To generate realistic queries, we have used the information gathered from the query logs of the BibFinder and NetBib sites. For both of these archives, results from queries are always a list of matching articles. Any refinement made by a user (possibly overwhelmed by a huge list of results) is independent of the previous query. We may then consider all queries as independent of each other. Both logs agree on the fact that queries are mainly made using author names, with conference and publication date as second and third criteria. We have therefore mapped our experimental query model directly to the structure and frequency of the queries in the BibFinder log, as shown in Figure 6.8.

Realistic popularity model

After modeling *how* articles are queried, we need to model *which* of them are most often requested. We have first observed author popularity, by counting the number of queries to each author in the BibFinder and NetBib log traces: the popularity of an author is defined as the probability of having a query with the "author" field being for that author. Similarly, we have computed the popularity of requested article titles in the BibFinder log. Finally, we have observed the probability that each of the top-10,000 articles in the CiteSeer database [145] gets cited; although this is a measure of how frequently a given article is cited, it can be considered as a clear indication of the article's popularity.

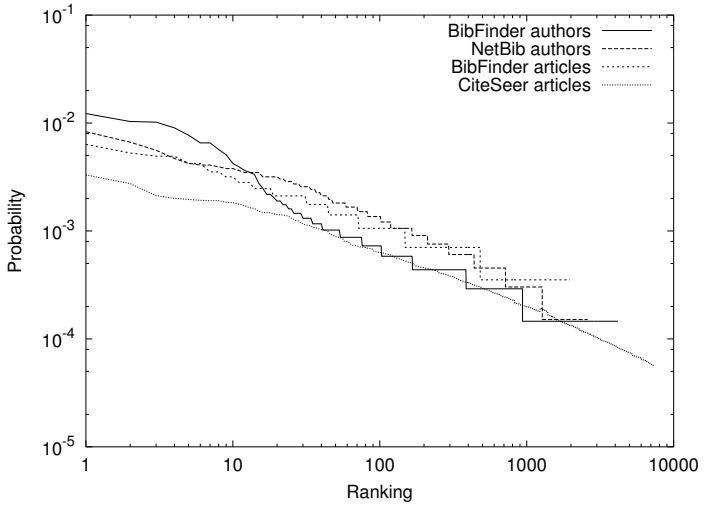


Figure 6.10: Popularity distribution for authors and titles present in NetBib, BibFinder, and CiteSeer.

The observed probability distribution are shown in Figure 6.10 (logarithmic scales, with articles ordered by decreasing rank of popularity). It appears clearly that all probabilities follow roughly a Zipf distribution [146, 100]. That is, the popularity of an article with rank i is $p_i = \frac{c}{i^b}$, where $c, b \in \mathbb{R}^+$ and $\sum p_i = 1$. A few articles appear in many queries, while most are seldomly requested.

To model article popularity for our simulations, we have computed (using the minimum square method) from the plot of BibFinder's author probabilities the line that best fits the distribution; switching to a linear scale, we obtain the power-law distribution describing the popularity of each article and the associated cumulative distribution function. In order to simplify the simulations, we have considered a limited collection of $10,000$ articles. After adapting the parameters of the power-law distribution to match the finite population of articles, we obtain a complementary cumulative distribution function of $\bar{F}(i) = 1 - F(i) = 1 - 0.063 \cdot i^{-0.3}$, where i is the ranking of the article. Numerical values are given for illustration purposes only: the behavior is defined by the *family* of the probability distribution function, i.e., a Zipf function. The distribution function is plotted in Figure 6.11.

From the figure we can appreciate that, because of the skewed nature of the distribution, using only $10,000$ articles does not change significantly the behavior of the model. The remaining articles from the original DBLP archive would be requested so rarely that we can effectively neglect their existence. When constructing the query workload for the simulation, we first choose an article according to the popularity distribution. Then, we select the structure of the query and assign the corresponding fields, according to the following probabilities:

- **Author**, with probability 0.6,
- **Title**, with probability 0.2,

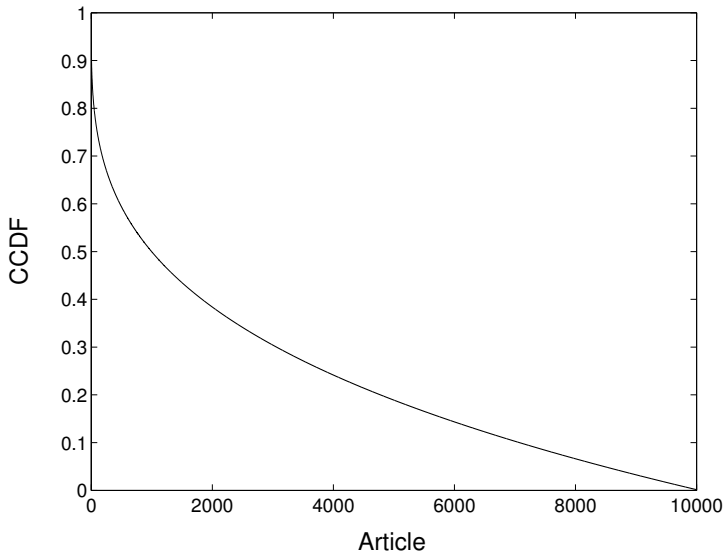


Figure 6.11: Complementary cumulative distribution function of the articles ranking.

- **Year**, with probability 0.1,
- **Author and Title**, 5% of the times, and
- **Author and Year**, another 5% of the times.

The resulting query is subsequently used as input to locate the article selected initially.

6.5.4 Caching

As previously discussed, some articles are much more popular than others, and hence will be requested more frequently. By creating cache entries (shortcuts) for these articles, we can probabilistically improve look-up times and reduce the load on the system. We have observed the effectiveness of the adaptive caching mechanism described in Section 6.4, in which peers create cache entries along the index look-up paths of successful queries. In the simulation, we study three different caching policies:

- *Multi-cache:* Shortcuts are created on each peer along the index look-up path. The size of the cache is unbounded.
- *Single-cache:* Shortcuts are created only on the first peer that was contacted. The size of the cache is unbounded.

- *LRU:* Similar to the single-cache policy, but only a limited number of shortcuts can be stored on each peer. When the cache is full, a least-recently used (LRU) replacement algorithm is used.

6.5.5 Simulation

Simulating P2P networks of different sizes is of no use for our experiments. The number of peers can affect the DHT look-up latency and the number of keys stored per peer, but does not impact the effectiveness of our indexing techniques. Any optimization of the underlying P2P network to reduce look-up latency will improve the response time when searching through the indices, but these are completely independent issues (layered protocols). Our experiments simulate a P2P system of 500 peers, on top of which a distributed bibliographic database of $10,000$ articles is implemented. For each of the indexing schemes and caching policies described previously, we measure different important metrics during simulation. Each simulation consists of sequentially feeding the indexing network with $50,000$ queries from our query generator. The studied parameters are: number of user-system interactions required to find data, traffic generated, efficiency of shortcuts, and storage dedicated to caching. In all the figures of this section, S, F, and C stand for *simple*, *flat*, and *complex* indexing, respectively. The LRU replacement caching policy is tested for an allowed maximum of 10, 20, and 30 cached keys per peer.

User-system interactions

A user sends a query and obtains as result a list of more specific queries (covered by the original query). The user then selects one query from the results that matches the target article. This process iterates until the article is found. Isolated from other concerns, a user would like to experience a minimal number of iterations to locate the desired data. One should note, however, that the number of iterations is expected to increase when the user initiates the search with a generic query; there is clearly a trade-off between the amount of information initially known about the searched data, and the number of steps necessary to locate that data.

Simulation results are shown in Figure 6.12. The *flat* indexing scheme, which creates the shortest query chains (see Figure 6.9), also requires the fewest interactions to locate data. Except for this flat scheme, caching further reduces the number of look-up steps, which becomes smaller as the cache size increases. The multi-cache policy is not shown here because it presents the same characteristics as the single-cache policy. Due to the rather simple representation of data, index chains are in general short. More complex data representations would need longer index chains, where the effect of shortcuts is more important.

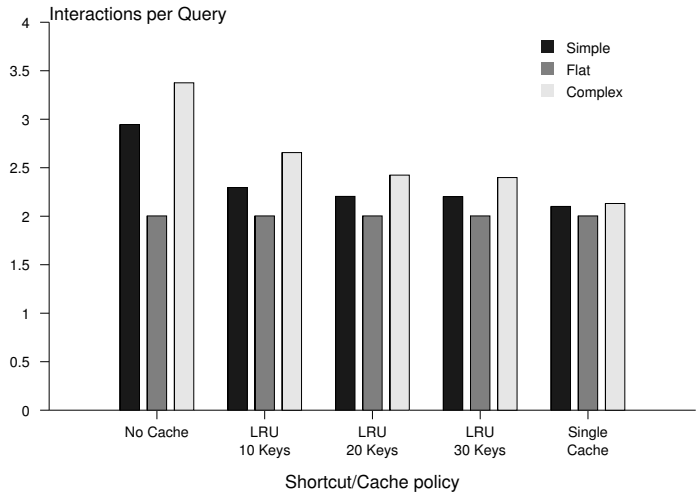


Figure 6.12: Average number of interactions required to find data.

Generated traffic

Like in any other network, it is desirable to generate as little traffic as possible to avoid network congestion and save system resources. Another interpretation of the traffic generated is the number of responses that a user receives for a query, i.e., the size of the result sets (traffic is mainly driven by responses, which usually outnumber a single query). The more traffic, the more responses a user gets for a query. Large results sets increase the burden of the user because responses are less relevant and more post-processing is required. Note that, since our search process is completely deterministic, more results do not imply that more information is available (as could be the case of an Internet search engine such as Google [147], or the results provided by an unstructured P2P network like Gnutella), but a *less precise answer*. The query and response traffic measured during the simulations is shown in Figure 6.13.

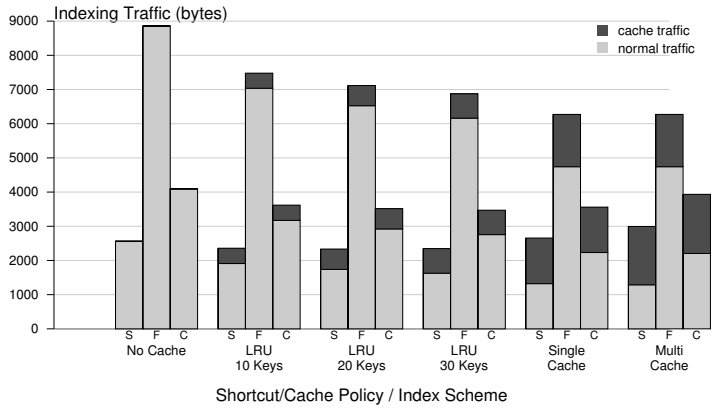


Figure 6.13: Average network traffic (bytes) generated per query.

Cache traffic, resulting from the creation of cache entries after successful look-ups, is shown in dark gray. It appears clearly that the *flat* indexing scheme generates much more traffic than any other. In fact, it does not allow for any indirection, and each query receives directly the descriptors of *all* articles that match the query, instead of a relevant set of more specific queries that allows refinement of the search process. We observe that the utilization of the cache saves network bandwidth. Unsurprisingly, larger cache sizes yield more cache traffic and less overall traffic transmitted over the network. The multi-cache policy leads to more cache traffic than single-cache, because the latter creates only one cache entry upon successful look-up. The network traffic of the flat scheme is reduced because, even if the number of steps to locate an item is the same, independently of the cache policy, the cache allows for direct retrieval of the file from the responsible peer, without DHT look-up, saving bandwidth.

Introducing peer failures in the experiment would require accounting for the DHT look-up traffic with a probability equal to that of peer failure when the cache is used. The expected look-up traffic should also be modified according to the peer failure probability, as was shown in Chapter 2.

Cache efficiency

We have observed the effectiveness of the adaptive distributed cache by measuring its hit ratio, that is, the fraction of requests that do not need to go through a full search process because the relevant data is already available in the cache.

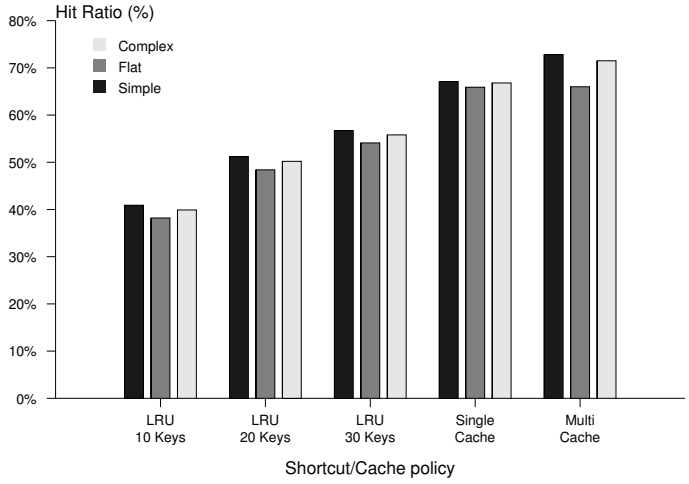


Figure 6.14: Cache efficiency: distributed hit ratio (percentage of queries found in the distributed cache).

Figure 6.14 shows the results for the different policies tested. It is interesting to note that the multi-cache policy is only marginally more efficient than the single-cache policy. Although

the multi-cache policy stores a cached key on every peer in the index chain followed to find the data, most cache hits occur in the first peer of the chain: 86% for the *simple* scheme, 99.9% for *flat*, and 84% for *complex* (the flat indexing does not get 100% cache hits at the first peer in the chain because some queries are not indexed, and thus require a second hop of generalization and specialization to get an actual cache hit). Indeed, in our user model queries are usually very simple and broad, thus directing the user to the beginning of an index chain. It is also worth noting that, when limiting the number of cache entries per peer to just 10, cache efficiency is still more than half that of policies with unbounded cache size.

Cache storage

The number of regular keys stored on each peer depends on the indexing scheme. We observed an average of 155 keys per peer for *simple*, 195 for *flat*, and 180 for *complex*. The overhead due to metadata is between 8 and 10 entries per data entry. But as we show before in Section 6.5.2, the metadata size is negligible when compared to that of the actual data. The storage dedicated to cached keys should remain relatively small, while allowing for an efficient distributed cache. Cache storage sizes after feeding the 50,000 queries are shown in Figure 6.15.

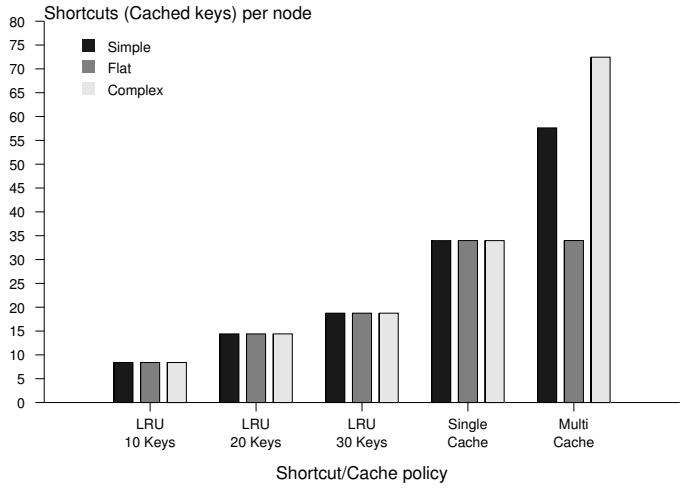


Figure 6.15: Average number of cached keys per peer.

As expected, the single-cache policy is approximately twice as space-efficient as the multi-cache policy, which creates more cache entries. The *flat* indexing scheme is unaffected because its index chains are so short that they only allow for a key to be cached in the first peer.

We also observed the maximum number of cache entries stored across all peers. For the multi-cache policy, we find up to 345 cached keys in a peer with the *simple* indexing. For the *flat* and *complex* indexing, the maxima are 253 and 413 keys, respectively. The single-cache policy uses a maximum of 253 cached keys, regardless of the indexing scheme used. For the

limited cache size policies, the maxima are obviously the cache capacities (10, 20, and 30). 72% of the caches were full after feeding the 50,000 queries with the LRU10 policy, 51.2% with LRU20, and 37.6% with LRU30, regardless of the indexing scheme. Overall, 4.4% of the caches were completely empty, with not a single entry.

As the cache management directly depends on the query workload, cache utilization is not uniform. The distributed cache improves look-up performance, but does not solve load-balancing issues in the overlay network.

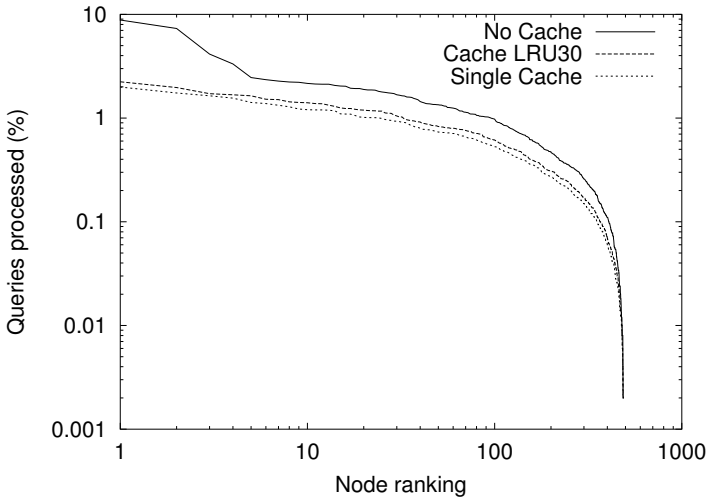


Figure 6.16: Percentage of queries processed by each peer in the network.

Hot-spots

As just discussed, the query patterns of the users lead to an unbalanced utilization of the distributed cache. We have studied the distribution of the processing of queries among peers. In Figure 6.16, we show the percentage of the 50,000 queries issued during the simulation that accessed each peer (for clarity, we only show the results for the *simple* indexing scheme). Note that they sum to more than 100% because each original query may generate other queries during the iterative look-up process. We observe that the busiest peer is affected by almost 1 out of every 10 queries. The five most loaded peers get a significant higher load. This can be motivated by the storage of indices corresponding to two or more very popular articles. For example, if two articles are very requested and happen to have been published in the same conference and the same year, the Conference+Year index for those two articles will experiment very high demand (actually more than each of the actual articles taken separately). Caching slightly improves the situation for the most severely stressed peers. We expect the load to be better balanced with larger content and query workloads. Note also that any optimization of the underlying P2P substrate for hot-spot avoidance (e.g., using replication) will apply to index accesses as well.

Locating non-indexed data

As previously mentioned in Section 6.4.2, it may happen that a user issues a query using a combination of fields (author, year, etc). that has not been indexed (in our case, as shown in Section 6.5.2, the Author+Year queries find no corresponding index in Figure 6.9). In that situation, the system can first generalize the original query to find a matching index entry, and then specialize it by following indices until the target data is located. In our experiments, this event happened approximately 2,500 times for all indexing schemes when no cache is used. When an error is encountered, one extra interaction is generally necessary to find a suitable index (two interactions in a few rare cases).

	Simple	Flat	Complex
No cache	2,513	2,513	2,513
LRU30	810	874	838
Single-cache	563	600	581

Table 6.1: Number of queries to non-indexed data.

Table 6.1 shows the number of accesses to non-indexed data, i.e., recoverable errors, as a function of the cache policy deployed in the system. We can observe that the cache reduces the number of errors, because an index entry is created automatically after the first look-up; subsequent queries from other users can locate the data using the cache entry, and hence do not experience an error. Our indexing techniques, together with the adaptive distributed cache, are thus very flexible: although the indexing scheme is generally chosen at deployment time, the system can still adapt to the user querying habits by creating shortcuts dynamically.

6.6 Conclusion

A major limitation of Peer-to-peer systems with DHT look-up is that they only support exact-match look-ups: one needs to know the exact key of a data item to locate the peer responsible for storing that item. Since peer-to-peer users tend to submit broad queries to look up data items, P2P systems using DHT look-up services need to be augmented with mechanisms for locating data using incomplete information.

In this chapter, we have proposed techniques for indexing the data stored in the peer-to-peer system. Indices are distributed across the peers of the network and contain key-to-key (or query-to-query) mappings. Given a broad query, a user can look up the more specific queries that match its original query; the look-up service can be recursively queried until the user finds the desired data items. As they can be layered on top of an arbitrary P2P network using a DHT look-up, our indexing techniques directly benefit from any mechanisms implemented in the DHT to deal with failures or hot-spot avoidance. We have performed a comprehensive evaluation to demonstrate the effectiveness of data indexing on a distributed P2P bibliographic database,

using different strategies and realistic user query workloads. Finally, we have proposed an adaptive caching mechanism that has proved to improve performance of accesses to popular content.

Chapter 7

Conclusions and perspectives

7.1 Conclusion

A conclusion is the place where you got tired of thinking.

Arthur McBride Bloch. Author of Murphy's Laws

Peer-to-peer systems for the people's PCs have been the latest guests to arrive to the Internet. They represent a new way of offering services at the edges of the Internet by users and for users. The potential of P2P networks to create new, innovative services is at the origin of their immense growth to reach millions of users. This is also the reason that make P2P systems worth considering for new Internet services to be developed. But Peer-to-peer's distributed nature, widespread usage, and lack of a centralized management of resources require algorithms that can smoothly scale to millions of users and use efficiently the resources of a very large number of machines connected through the Internet.

Structured P2P networks by means of Distributed Hash Tables present interesting properties (scalability to a large population of users, fast look-up of resources) for Content Distribution and Content Location. We have explored these issues by first presenting an introductory study on hierarchical DHTs that has served as a base for the development of our own DHT, called TOPLUS. The appropriate features of TOPLUS have enabled us to propose an application-level multicast P2P scheme we have called MULTI+. Its characteristics are very well suited for large-scale distributed scenarios requiring highly efficient resources usage. Finally, we have addressed resource location by storing indexes in the P2P network that can be searched through a DHT look-up service.

We now summarize the main contributions of this thesis:

- Inspired by the somewhat hierarchical organization of Internet routing, we have proposed a generic framework for the hierarchical organization of peer-to-peer overlay networks, and we have demonstrated the many advantages it offers over a flat organization. A hierarchical design offers higher stability by using more "reliable" peers (superpeers) at the top levels. It can use various inter- and intra-group look-up algorithms simultaneously, and treats join/leave events and key migration as local events that affect only a single group. By organizing peers into groups based on topological proximity, a hierarchical organization also generates fewer messages in the wide area and can significantly improve the look-up performance, as well as cache popular content in local groups. By first querying the responsible peer within one's own group, popular objects are dynamically pulled into the various groups. This local-group caching can dramatically reduce download delays in peer-to-peer file-sharing systems. We have introduced minor adaptations in the Chord look-up algorithm, in order to obtain a Chord-based hierarchical DHT. This introduces a selective redundancy in the routing tables of the most stable peers that makes the Chord look-up algorithm more robust against the inherent instability of P2P networks. Using a novel approach, we have quantified the improvement in the performance of the DHT look-up using the hierarchical Chord. When each peer is down with a certain probability, a hierarchical organization reduces the length of the look-up path dramatically for the case where the failure probability of superpeers is smaller than the failure probability of regular peers. Thus we provide a way to improve the performance of structured P2P networks through a hierarchy, in a similar fashion to the one that drives nowadays P2P file-sharing networks in the Internet towards a hierarchical structure.

- TOPLUS is a fully distributed and symmetric DHT that integrates topological considerations in its design. As a hierarchical DHT, peers are organized into groups. We map the groups directly to the underlying topology, resulting in an unbalanced tree without a rigid partitioning. This is the first approach, to our knowledge, to a DHT providing locality to P2P applications without taking network measurements, just by using a namespace highly correlated with the Internet IP addressing space. TOPLUS also provides a routing scheme that makes big (in terms of latency) jumps first rather than small ones as Pastry does. We have shown that TOPLUS offers excellent stretch properties, resulting in a very fast look-up service. Although TOPLUS suffers from some limitations, which we have exposed and discussed, we believe that its fast look-up and its simplicity make it a promising candidate for large-scale deployment in the Internet.

- MULTI+ is a method to build application-level multicast trees on P2P systems. MULTI+ relies on TOPLUS in order to find a proper parent for a peer in the multicast tree. MULTI+ exhibits the advantage of being able to create topology-aware content distribution infrastructures without introducing extra traffic for active measurement. The proximity-aware parent selection scheme improves the end-to-end latency. Using host coordinates that are calculated offline and obtained at join time (as is done for TOPLUS) completely avoids the need for any measurement. MULTI+ also decreases the number of redundant flows

that must traverse a given network, even when only few connections per peer are possible, which allows for a more efficient bandwidth utilization. We have also presented a practical approximation to the full TOPLUS tree: the Hierarchical Internet Partition has been shown as a good compromise, providing sufficient accuracy but simplifying a lot the overlay construction. MULTI+ remains robust despite massive peer failure and is able to maintain low end-to-end delays even in realistic situations where the number of connections per peer is limited.

- We have proposed techniques for indexing the data stored in DHT-based peer-to-peer networks. A major limitation of peer-to-peer systems using a DHT to look-up resources is that they only support exact-match look-ups: one needs to know the exact key of a data item to locate the peer responsible for storing that item. Since peer-to-peer users tend to submit broad queries to look up data items, DHT peer-to-peer systems need to be augmented with mechanisms for locating data using incomplete information. Indexes are distributed across the peers of the network and contain key-to-key (or query-to-query) mappings. Given a general query, a user can look up the more specific queries that match its original query; the DHT can be recursively queried until the user finds the desired data items. Being seamlessly integrated in the P2P system, our indexing techniques directly benefit from any mechanisms implemented in the DHT to deal with failures or hot-spot avoidance. We have performed a comprehensive evaluation to demonstrate the effectiveness of data indexing on a distributed P2P bibliographic database, using different strategies and realistic user query workloads. Finally, we have proposed an adaptive caching mechanism that that improves the performance of accesses to popular content. We have seamlessly introduced limited querying and searching capabilities into structured P2P networks, adding some flexibility to the otherwise very efficient DHT look-up.

7.2 Perspectives

Peer-to-peer systems is a growing field where new applications and algorithms, as well as new issues and challenges, appear every day. From our experience during the course of this thesis, we present here the topics that we have left open for future research:

- We have seen that DHTs map a key to a single peer in a P2P network, but we have noticed that, depending on the P2P application, keys are requested with very different frequency. This introduces important unbalance among the load of peers (Section 6.5), independently of the distribution of keys over peers in the P2P system. New algorithms are needed for DHT-based P2P system in order to intelligently react to the users behavior. As we have previously said, users preferences cannot be inferred *a priori*, and may change over time. We propose that peers experiencing heavy load reactively replicate frequently requested keys on other randomly selected peers. The peers under heavy load would need to temporary modify the routing tables of their neighbors in order to reroute queries for keys with high demand towards their new locations. If the load on the neighbors

themselves is still high, they may proceed in the same manner. This would lead to a redistribution of queries among a set of randomly chosen peers storing the high-demand keys. The modifications on the routing tables would remain as long as the load perceived by peers exceeds some threshold. This policy rises security issues, as to which extent a peer can modify others' routing tables. Malicious peers may use this technique to hinder access to certain keys.

- Security in peer-to-peer systems is a field of enormous interest. There are many applications that may never profit from the possibilities of P2P networks unless distributed security schemes are developed. As was pointed out in the introduction of this thesis, trust among peers, collaboration enforcement and data integrity, among others, are necessary features for certain P2P applications (e-commerce, storage systems, etc). Designing fully distributed algorithms for basic security services is a very challenging task.

- The fact that keys in hierarchical DHTs are mapped to groups (and then to a peer inside the group, but a key can be replicated on multiple peers inside a group) present the advantage of groups being far more stable than peers. However, the issues raised by the failure or modification of groups (even if they seldom occur) are far more involved than those of a single peer. Moving all the keys from one group to another can trigger an avalanche of data traffic that might congest the network links. We have presented some lazy algorithms that quietly update the P2P network when new groups are created (Section 3.3), but these approaches may not be reactive enough to allow access to keys when network failures occur. We need algorithms that allow the "organic" development of structured P2P networks.

- Load balancing techniques among groups have been sketched in Section 3.4. However, we need a deeper analysis and evaluation of these and other approaches, like those based on linear hash functions [148]. Indeed, the problem of the lack of correlation between the peer population in a group and the number of keys mapped to that group needs to be further studied in the hierarchical DHT framework.

- While our data indexing techniques permit to look up data based on incomplete information, they still depend on the exact matching facilities of the underlying DHT. "Fuzzy" matching techniques offer interesting research perspectives for dealing with misspelled data descriptors or queries. Misspellings can also often be taken care of by validating descriptors and queries against databases that store known file descriptors, such as CDDB [149] for music files.

Summarizing these issues, the final question may be: can we find a solution conciliating the look-up speed of DHTs and the flexibility of unstructured P2P networks?

Appendix A

Coordinate Space Testing

A.1 Tang-Crovella Coordinate Space

Some methods ([82, 124, 150, 125]) to map hosts into a n-dimensional coordinate space have been developed. The main advantage is that given a list of hosts, their coordinates can be actively measured in $O(n)$ time (the distances of the hosts to a set of *landmark* hosts). Then, the inter-host distance matrix can be calculated off-line. We use the coordinate space from the authors of [125] at Boston University, that we call TC (from the authors, Tang and Crovella) Coordinate Space.

Recently other, more distributed approaches have appeared in the literature. Methods like [123, 151] differ only in how coordinates for a host are obtained. Instead of using a fixed set of landmark nodes to calculate every host's coordinates, each machine can choose an arbitrary set of hosts for the same purpose. Our aim is to use a given coordinate space to study the topological characteristics of a given DHT or the properties of a topology-aware application. How are these coordinates obtained is not an issue in our experiments. However, should one of these methods be used to improve the presented P2P systems, a distributed approach to coordinate obtention would be preferred.

First of all we attempt to measure the degree of correlation between distances calculated from this coordinate space and the real measured distances for some random samples. Calculating distances in the coordinate space with different metrics gives different results. We choose as our working metric the one whose resulting distances represent real delays with more accuracy. But we first need a set of machines which we have the coordinates for, and also accurate measurements of the delays from each machine to the others.

RIPE has deployed a number of machines (probes) that measure the delay among each other. We use the median value of each of these delays measured during the last week as the distance between their two corresponding machines. We find the IP address of one RIPE TTM probe in

the TC Coordinate space. We find 15 other probes in the 26 bit prefix neighborhood of some IP address in the TC Coordinate space. We assume that the delay for the probes is similar to the hosts with IP addresses in the same 26 bit prefix, because each of those prefixes represents a network of 64 machines.

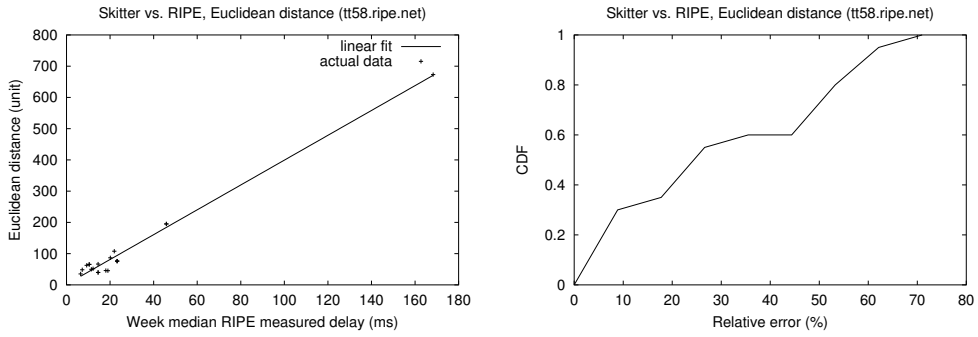
We compare actual delay to D_+ and Euclidean distances. We define the D_+ metric as in [127]:

$$D_+(x_i, x_j) = \min_{k=1,\ldots,M}(x_{ik} + x_{jk})$$

and the Euclidean metric as:

$$D_{Euclidean}(x_i, x_k) = \sqrt{\sum_{k=1,\ldots,M}(x_{ik} - x_{jk})^2}$$

for any two hosts identified by their M-coordinate vector x_i and x_j.

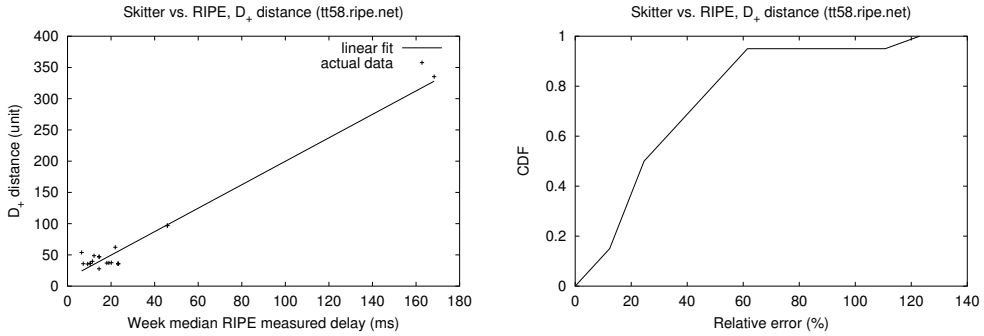


(a) Linear fit between calculated and measured distances.

(b) Relative error to linear fit data.

Figure A.1: Euclidean distances compared to measured delay.

If the real delays and the calculated distances are well correlated, they should fit a line, with delays on one axis and calculated distances on the other. Obviously, we do not expect the coordinate space to provide directly the delay between two machines: but if one delay is twice another, their corresponding distances in the coordinate space should keep similar proportionality. The plots in Figure A.2 show that both metrics are able to accurately calculate the delays among the RIPE probes. We also plot the relative error of the calculated distances to the linear fit. The rare distances that present an important relative error are usually pretty small. This means that the present error in the calculated distances is not changing the perception of real ones: hosts that are close are perceived as close and those far away remain distant. The authors of [127] claim that D_+ obtains better results, but it is not our case. We have thus chosen



(a) Linear fit between calculated and measured distances.

(b) Relative error to linear fit data.

Figure A.2: D_+ distances compared to measured delay.

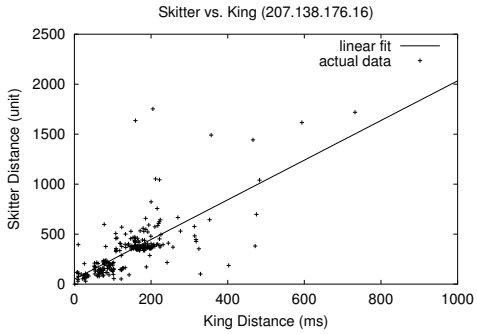
the Euclidean distance for our experiments. The RIPE probes should be giving good stable measurements, while other sets may present less accurate results.

We would like to test the TC Coordinate Space with more widely distributed hosts. The RIPE probes provide a set of accurate delays, but they are well connected hosts located in Europe. We proceed to compare the distances obtained among 260 random hosts in the TC Coordinate Space with the distances given by measurements with `King`. We also measure the relative error of the distances obtained from the coordinates to a linear fit of those distances and the ones obtained with `King`.

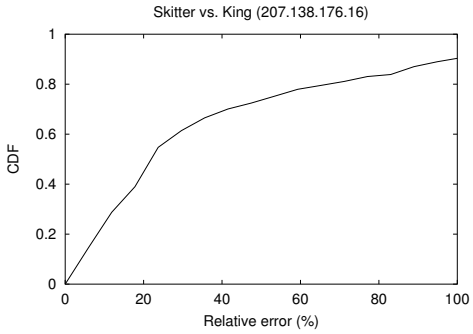
We measured distances to hosts in the US, Europe, Far East Asia and Australia. The locations were obtained with *NetGeo* [152]. Perhaps the relative error could be improved through weighing samples according to some "subjective" perception. Summarizing, we may say that for most US and European locations, 60% of the calculated distances present at most a 20% relative error. Considering the Far East targets, performance for the same fraction of the distances degrades reaching up to 30-40% relative error. The Australian locations perform similarly. It must however be reminded that we are comparing the distances calculated in the TC Coordinate Space with those measured with `King`, which are only an estimation of the real delays.

A.1.1 USA Hosts

Most USA locations present low relative error, and we can thus consider that distances calculated with the TC Coordinate Space are accurate enough. As most of the IP space belong to the USA, these results reinforce the general validity of this method to measure distances between random hosts. We plot below sample measurements for IP addresses 207.138.176.16, 66.71.191.66 and 4.19.161.2.

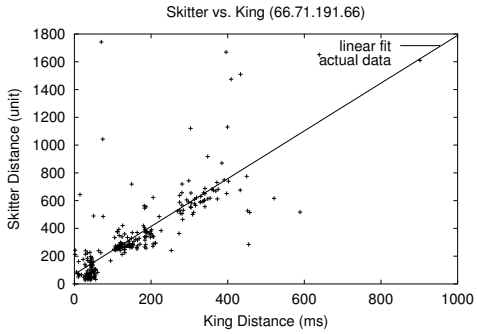


(a) Linear fit between calculated and measured distances.

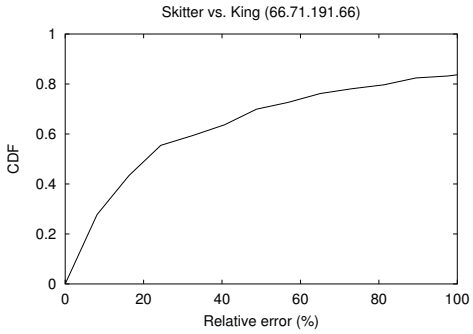


(b) Relative error to linear fit data.

Figure A.3: USA: Distances to 207.138.176.16

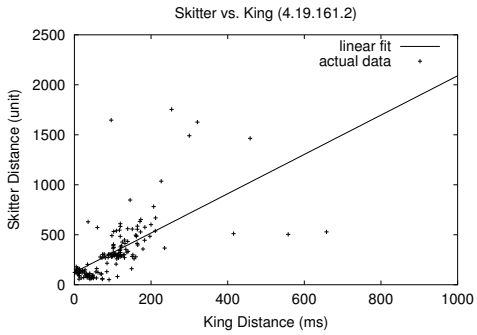


(a) Linear fit between calculated and measured distances.

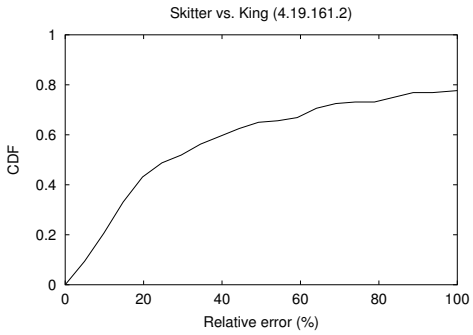


(b) Relative error to linear fit data.

Figure A.4: USA: Distances to 66.71.191.66



(a) Linear fit between calculated and measured distances.

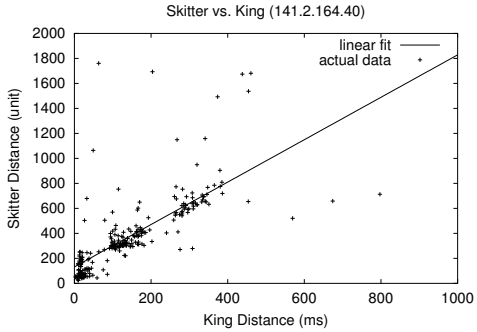


(b) Relative error to linear fit data.

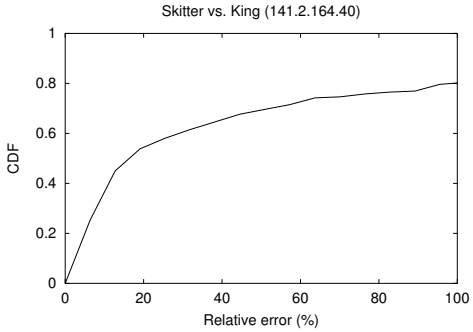
Figure A.5: USA: Distances to 4.19.161.2

A.1.2 European Hosts

The distances calculated for European locations give similar performance to those in the US. We present examples for a host in Germany (141.2.164.40), Poland (213.76.162.177) and UK (138.37.36.200).

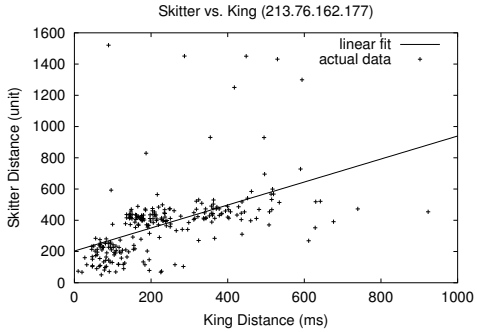


(a) Linear fit between TC/real distances.

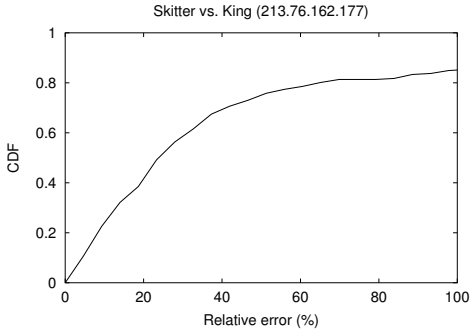


(b) Relative error to linear fit data.

Figure A.6: Europe: Distances to 141.2.164.40 (DE)

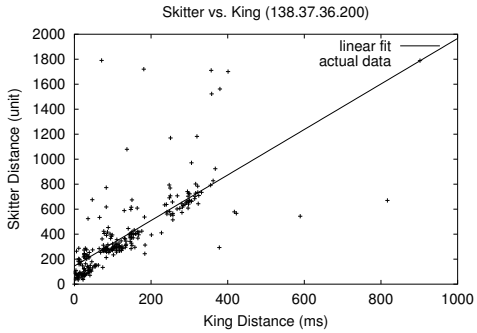


(a) Linear fit between TC/real distances.

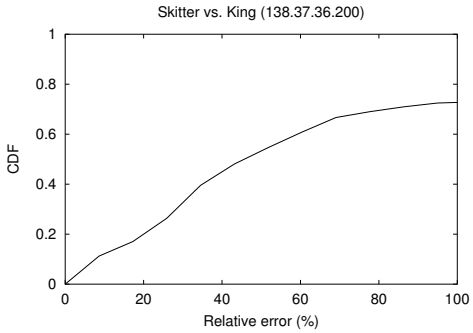


(b) Relative error to linear fit data.

Figure A.7: Europe: Distances to 213.76.162.177 (PL)



(a) Linear fit between TC/real distances.

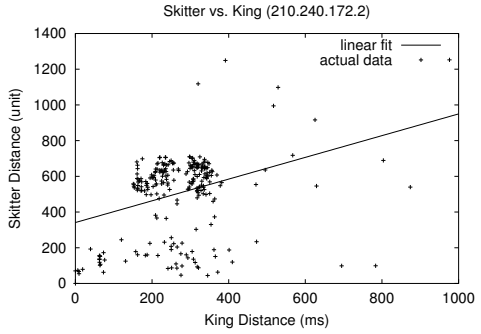


(b) Relative error to linear fit data.

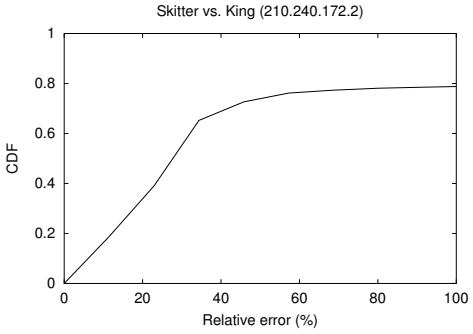
Figure A.8: Europe: Distances to 138.37.36.200 (UK)

A.1.3 Far East Hosts

The TC Coordinate Space performs worse for the Far East locations than for the previous ones. This may be motivated by the fact that the coordinate space is obtained from measurements to Skitter hosts, which are mainly located in the US and Europe. We present two examples, in Taiwan (210.240.172.2) and South Korea (211.174.60.101).

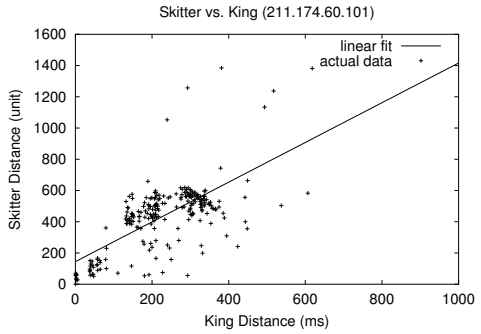


(a) Linear fit between calculated and measured distances.

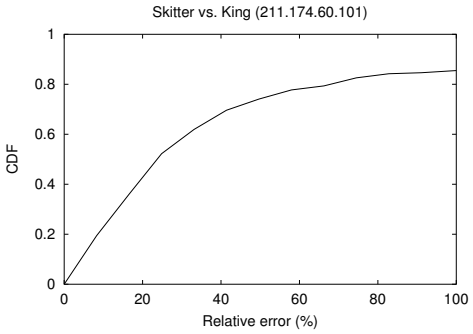


(b) Relative error to linear fit data.

Figure A.9: Far East: Distances to 210.240.172.2 (TW)



(a) Linear fit between calculated and measured distances.

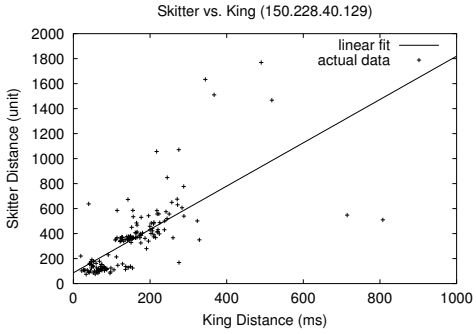


(b) Relative error to linear fit data.

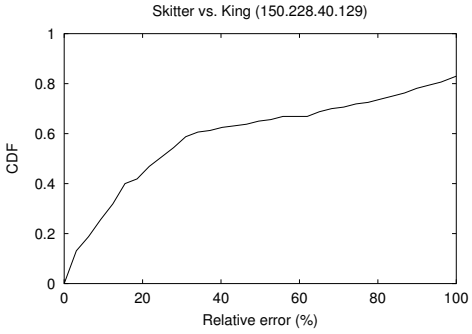
Figure A.10: Far East: Distances to 211.174.60.101 (KR)

A.1.4 Australian Hosts

For the same reasons that make the TC Coordinate Space not work very well for Far East hosts, distances to locations in Australia are calculated with a significant relative error.

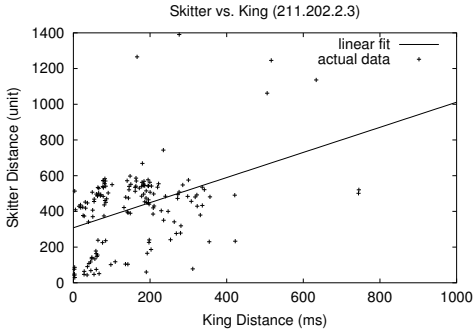


(a) Linear fit between calculated and measured distances.

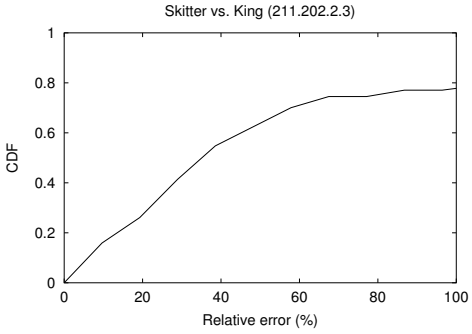


(b) Relative error to linear fit data.

Figure A.11: Australia: Distances to 150.228.40.129



(a) Linear fit between calculated and measured distances.



(b) Relative error to linear fit data.

Figure A.12: Australia: Distances to 211.202.2.3

Appendix B

Multicast Tree of One Thousand Peers

For completeness, we present in this appendix the results of the benchmarks performed on MULTI+ with a population of 1,000 peers. The main purpose is to show how the performance of MULTI+ gets better the larger the peer population, by comparing these results with those in Section 4.5.3.

In this experiment we test the characteristics of multicast trees built with MULTI+ using a set of 1,000 peers from the TC address space. We use the TC coordinate space to measure the distance between every pair of hosts.

The high-level characteristic of the TOPLUS tree obtained is as follows: 47 tier-1 groups, 781 inner-groups and 1,000 peers.

In the test we measure and plot the same parameters as those in Section 4.5.3, using their CDF (Cumulative Distribution Function):

- The percentage of peers that connect to the closest peer in the system when they arrive (Figure B.1).
- The percentage of the peers in the total system, when the full multicast tree is built, closer to one peer that this peer's parent. Those figures *exclude* the peers directly connected to the source (Figure B.2).
- The fan-out or out-degree of peers in the multicast tree. Obviously leafs in the tree have 0 fan-out.
- The level peers occupy in the multicast tree. The more levels in the multicast tree, the more delay we incur in along the transmission path and the more the transmission becomes subject to losses due to peer failure (Figure B.3).
- The latency from the root of the multicast tree to each receiving peer (Figure B.4).

- The number of multicast flows that go into and out of each TOPLUS group (network) (Figure B.5).

	Fan-out limit							
	2	3	4	5	6	7	8	∞
to closest peer upon arrival	5.2	5.3	6.7	7.0	7.9	8.0	7.9	8.4
to closest peer in full system	3.5	2.0	4.4	4.9	5.2	5.6	5.5	6.1

Table B.1: FIFO parent selection: Percentage of peers connected to closest peer, depending on the maximum number of connection allowed.

	Fan-out limit							
	2	3	4	5	6	7	8	∞
to closest peer upon arrival	10.7	12.1	12.3	12.0	12.9	12.0	12.3	8.4
to closest peer in full system	5.9	7.6	7.6	7.7	8.8	7.2	7.5	6.1

Table B.2: Proximity-aware parent selection: Percentage of peers connected to closest peer, depending on the maximum number of connection allowed.

We show in Table B.3 the maximum number of flows going through a TOPLUS group interface. MULTI+ reduces the average and maximum number of flows *per network*, when compared to an arbitrary connection scheme and assuming each TOPLUS group (IP network prefix) represents a physical network.

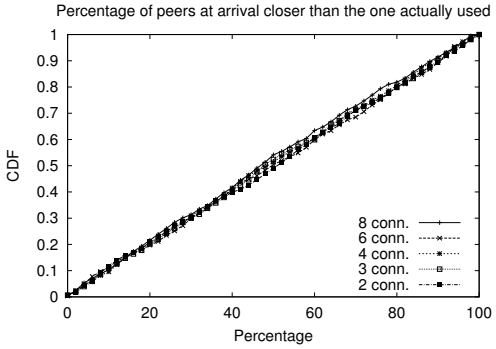
	Fan-out limit						
	2	3	4	5	6	7	8
FIFO	72	50	46	47	47	42	34
Proximity-aware	45	50	37	27	26	23	18
Random	184	186	197	176	177	191	186

Table B.3: Multicast Tree of 1,000 Peers: Maximum number of flows through a group interface.

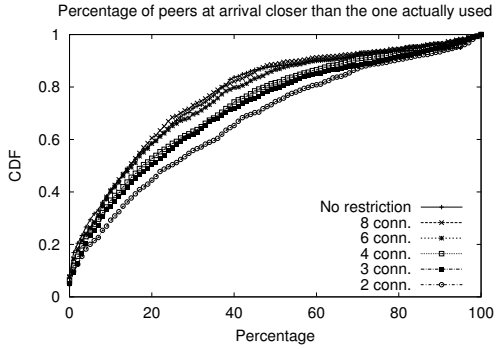
In average, each interface sees 2 flows with MULTI+ (independently of the parent selection policy), while 4 are used with the random algorithm.

	Fan-out limit			
	2	4	6	8
FIFO	$2,951(\pm 90)$	$1,706(\pm 71)$	$1,339(\pm 62)$	$1,121(\pm 60)$
Proximity-aware	$2,319(\pm 68)$	$1,109(\pm 52)$	$674(\pm 41)$	$568(\pm 36)$
Random	$3,493(\pm 106)$	$1,809(\pm 57)$	$1,337(\pm 51)$	$1,099(\pm 49)$

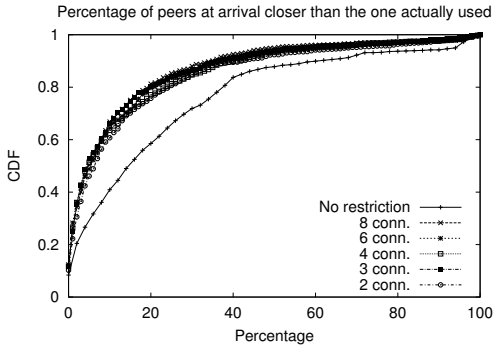
Table B.4: Multicast Tree of 1,000 Peers: Average delay from root to leaf.



(a) Random parent selection.



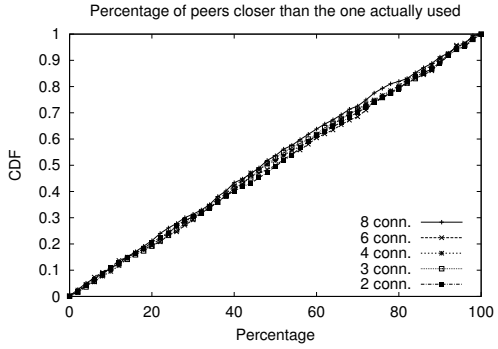
(b) FIFO parent selection.



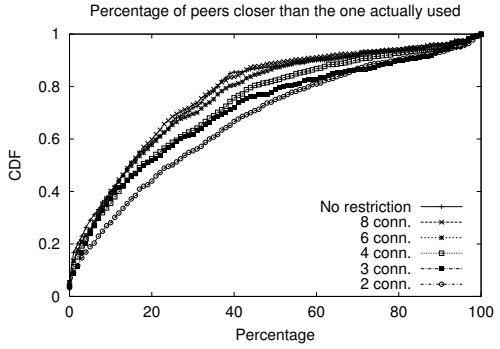
(c) Proximity-aware parent selection.

Figure B.1: Percentage of peers found upon arrival closer than the one actually used (for those not connected to the source).

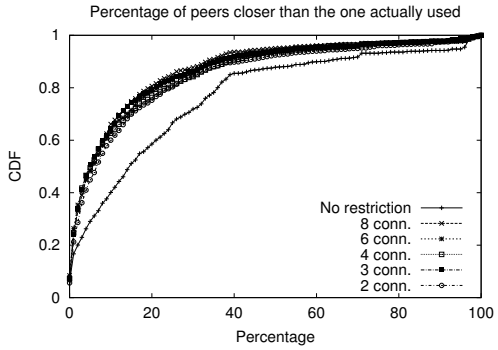
The average latency from the root of the Multicast tree to each destination is smaller for the proximity-aware parent selection algorithm (Table B.4, 95% confidence intervals). The more we limit the number of connections a peer can give service to, the better that MULTI+ behaves compared to an arbitrary parent selection algorithm.



(a) Random parent selection.

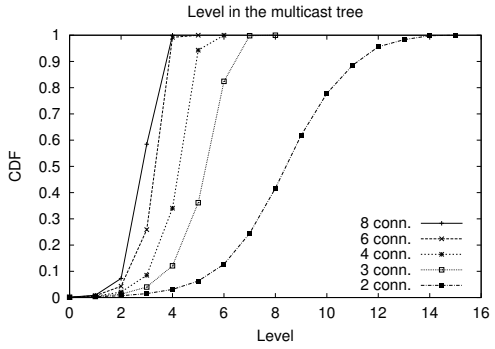


(b) FIFO parent selection.

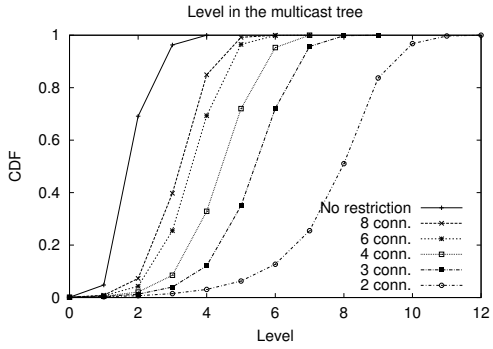


(c) Proximity-aware parent selection.

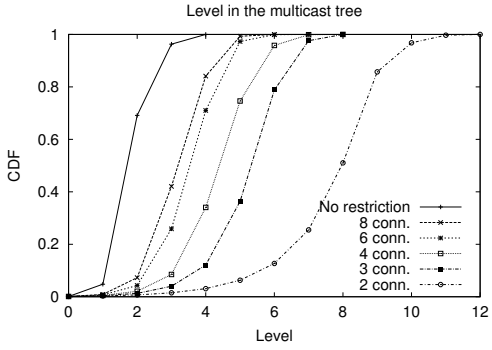
Figure B.2: Percentage of peers in the whole system closer than the one actually used (for those not connected to the source).



(a) Random parent selection.

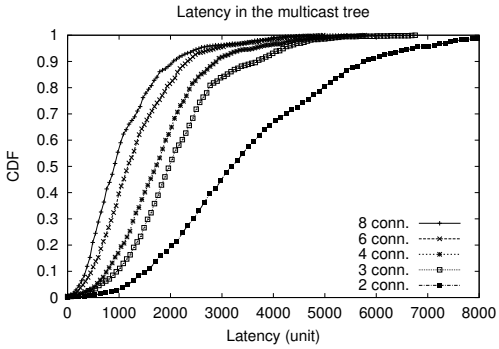


(b) FIFO parent selection.

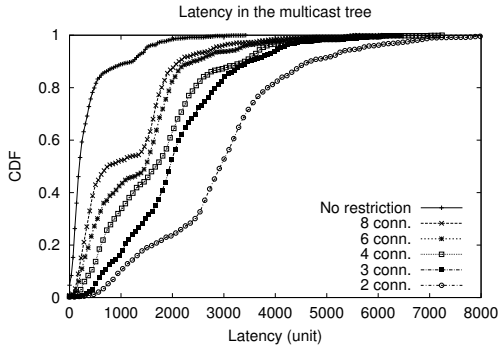


(c) Proximity-aware parent selection.

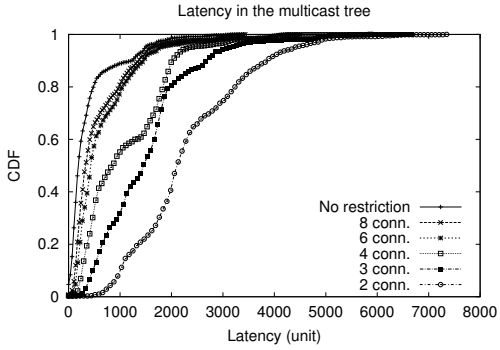
Figure B.3: Distribution of peers across levels in the Multicast tree.



(a) Random parent selection.

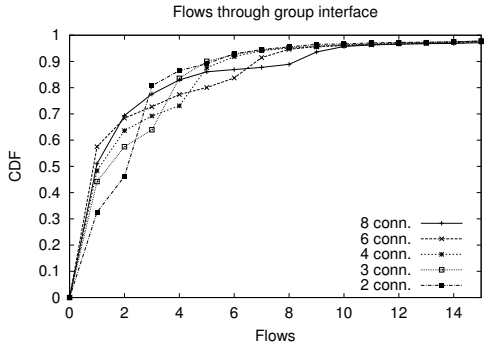


(b) FIFO parent selection.

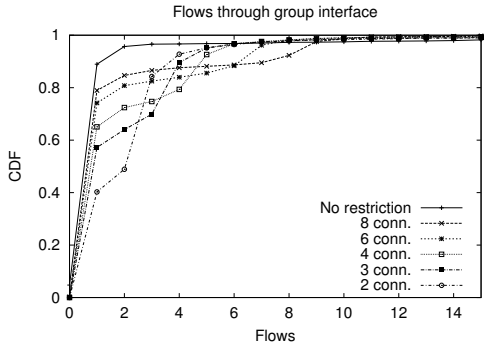


(c) Proximity-aware parent selection.

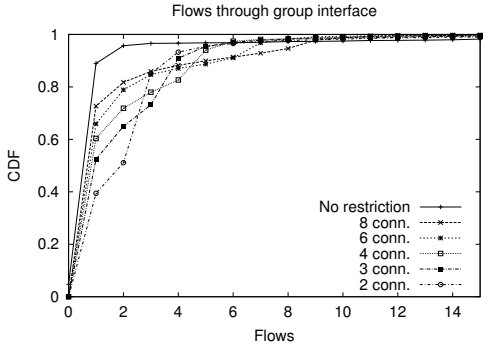
Figure B.4: Latency from root to leaf (in TC coordinate units) in the Multicast tree.



(a) Random parent selection.



(b) FIFO parent selection.



(c) Proximity-aware parent selection.

Figure B.5: Number of flows through group interface.

Bibliography

[1] F. Berman, A. J. Hey, and G. Fox, *Grid Computing: Making The Global Infrastructure a Reality*, ch. The Evolution of the Grid, pp. 65–100. John Wiley & Sons, 2004.

[2] Knowledge@Wharton, "The mega-media business model: Doomed to fail, or just ahead of its time?," *Research at Penn*, July 2002.

[3] J. Postel, "Internet Protocol." `ftp://ftp.rfc-editor.org/in-notes/std/std5.txt`, September 1981. RFC 791, STD 005.

[4] V. G. Cerf, "On the evolution of Internet technologies," *Proceedings of the IEEE*, vol. 92, pp. 1360–1370, September 2004.

[5] A. M. Odlyzko, "The stupid network: Essential yet unattainable," *ACM netWorker*, vol. 3, pp. 36–37, December 1999.

[6] R. C. Dixon and D. A. Pitt, "Addressing, bridging and source routing," *IEEE Network*, vol. 2, pp. 25–32, January 1988.

[7] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to end arguments in system design," *ACM Transactions on Computer Systems*, vol. 2, pp. 277–288, November 1984.

[8] A. M. Odlyzko, "Internet pricing and the history of communications," *Computer Networks*, vol. 36, pp. 493–517, 2001.

[9] J. Klensin, "Simple Mail Transfer Protocol." `ftp://ftp.rfc-editor.org/in-notes/rfc2821.txt`, April 2001. RFC 2821.

[10] B. Kantor and P. Lapsley, "Network News Transfer Protocol." `ftp://ftp.rfc-editor.org/in-notes/rfc977.txt`, February 1986. RFC 997.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1." `ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt`, June 1999. RFC 2616.

[12] J. Postel and J. Reynolds, "File Transfer Protocol." `ftp://ftp.rfc-editor.org/in-notes/rfc959.txt`, October 1985. RFC 959, STD 009.

[13] K. G. Coffman and A. M. Odlyzko, "Internet growth: Is there a "Moore's law" for data traffic?," in *Handbook of Massive Data Sets* (J. Abello, P. M. Pardalos, and M. G. C. Resende, eds.), Kluwer, 2001.

[14] A. M. Odlyzko, "Internet traffic growth: Sources and implications," in *Proceedings of SPIE, Optical Transmission Systems and Equipment for WDM Networking II*, vol. 5247, pp. 1–15, September 2003.

[15] M. Day, J. Rosenberg, and H. Sugano, "A model for presence and Instant Messaging." `ftp://ftp.rfc-editor.org/in-notes/rfc2778.txt`, February 2000. RFC 2778.

[16] D. Eastlake and T. Goldstein, "ECML v1.1: Field Specifications for E-Commerce." `ftp://ftp.rfc-editor.org/in-notes/rfc3106.txt`, April 2001. RFC 3106.

[17] A. M. Odlyzko, "The many paradoxes of broadband," *First Monday*, vol. 8, September 2003. `http://firstmonday.org/issues/issue8_9/odlyzko/index.html`.

[18] Gnutella. `http://gnutella.wego.com/`

[19] Napster. `http://www.napster.com/`

[20] KaZaA. `http://www.kazaa.com/`

[21] E-Mule. `http://www.emule-project.net/`

[22] BitTorrent. `http://bittorrent.com/`

[23] J. Klensin, "Role of the Domain Name System (DNS)." `ftp://ftp.rfc-editor.org/in-notes/rfc3467.txt`, February 2003. RFC 3467.

[24] "The O'Reilly Peer-to-peer and Web Services conference." `http://conferences.oreillynet.com/p2p/`, November 2001.

[25] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the Gnutella network: Properties of large-scale Peer-to-peer systems and implications for system design," *IEEE Internet Computing Journal*, vol. 6, January 2002.

[26] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of Peer-to-peer file sharing systems," in *Proceedings of Multimedia Computing and Networking (MMCN)*, (San Jose, CA, USA), January 2002.

[27] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-like P2P systems scalable," in *Proceedings of SIGCOMM*, (Karlsruhe, Germany), pp. 407–418, ACM, August 2003.

[28] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs," in *Proceedings of SIGMETRICS*, (Santa Clara, CA, USA), pp. 34–43, June 2000.

[29] E. Adar and B. A. Huberman, "Free riding on Gnutella," *First Monday*, vol. 5, October 2000.

[30] G. Caronni and M. Waldvogel, "Establishing trust in distributed storage providers," in *Proceedings of Peer-to-Peer Computing*, (Linköping, Sweden), pp. 128–133, IEEE Computer Society, September 2003.

[31] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP)*, (Banff, Alberta, Canada), pp. 131–145, ACM, October 2001.

[32] J. Xu, "On the fundamental tradeoffs between routing table size and network diameter in Peer-to-peer networks rks," in *Proceedings of INFOCOM*, (San Franciso, CA, USA), IEEE, March 2003.

[33] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-to-peer lookup service for Internet applications," in *Proceedings of SIGCOMM*, (San Diego, CA, USA), pp. 149–160, ACM Press, 2001.

[34] E. Cohen, A. Fiat, and H. Kaplan, "Associative search in Peer-to-peer networks: Harnessing latent semantics," in *Proceedings of INFOCOM*, (San Francisco, CA, USA), IEEE, April 2003.

[35] L. A. Adamic, R. M. Kurose, A. R. Puniyani, and B. A. Huberman, "Search in Power-Law networks," *Physical Review E*, vol. 64, 2001.

[36] J. Kleinberg, "The Small-World phenomenon: An algorithmic perspective," in *Proceedings of the 32nd ACM Symposium on Theory of Computing*, (Portland, OR, USA), pp. 163–170, ACM, May 2000.

[37] "The Gnutella protocol specification v0.41." `http://www.clip2.com/GnutellaProtocol04.pdf`.

[38] FIPS 180-1, "Secure Hash standard." `http://www.itl.nist.gov/fipspubs/fip180-1.htm`, April 1995.

[39] Dictionary of Algorithms and Data Structures – NIST, "Hash table." `http://www.nist.gov/dads/HTML/hashtab.html`, 2004.

[40] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of SIGCOMM*, (San Diego, CA, USA), pp. 161–172, ACM, 2001.

[41] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale Peer-to-peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)* (R. Guerraoui, ed.), vol. 2218 of *LNCS*, (Heidelberg, Germany), pp. 329–350, Springer, November 2001.

[42] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Tech. Rep. UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, April 2001.

[43] K. Aberer, "P-Grid: A self-organizing access structure for P2P information systems," in *Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS)*, vol. 2172 of *LNCS*, (Trento, Italy), Springer, September 2001.

[44] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. 34, pp. 892–901, October 1985.

[45] K. Lougheed and Y. Rekhter, "A Border Gateway Protocol 3 (bgp-3)." `ftp://ftp.rfc-editor.org/in-notes/rfc1267.txt`, October 1991. RFC 1267.

[46] A. J. Menezes, P. C. V. Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, October 1996.

[47] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Wiley, 2nd edition, October 1995.

[48] Y. Desmedt, "Threshold Cryptography," in *Proceedings of the 3rd Symposium on: State and Progress of Research in Cryptography* (W. Wolfowicz, ed.), (Rome, Italy), pp. 110–122, February 1993.

[49] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The EigenTrust algorithm for Reputation management in P2P networks," in *Proceedings of the 12th International World Wide Web Conference (WWW)*, (Budapest, Hungary), pp. 640–651, ACM Press, May 2003.

[50] J. R. Douceur, "The Sybil attack," in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)* (P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, eds.), vol. 2429 of *LNCS*, (Cambridge, MA, USA), pp. 251–260, Springer, March 2002.

[51] A. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure Overlay Services," in *Proceedings of SIGCOMM*, (Pittsburgh, PA, USA), pp. 61–72, ACM, August 2002.

[52] A. Fiat and J. Saia, "Censorship resistant peer-to-peer content addressable networks," in *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, (San Francisco, CA, USA), pp. 94–103, January 2002.

[53] M. Adler, R. Kumar, K. Ross, D. Rubenstein, D. Turner, and D. Yao, "Optimal peer selection in a free-market peer-resource economy," in *Proceedings of the 2nd Workshop on Economics of Peer-to-Peer Systems*, (Harvard University, USA), June 2004.

[54] K. Tamilmani, V. Pai, and A. E. Mohr, "SWIFT: A system with incentives for trading," in *Proceedings of the 2nd Workshop on Economics of Peer-to-Peer Systems*, (Harvard University, USA), June 2004.

[55] Z. Abrams, R. McGrew, and S. Plotkin, "Keeping peers honest in EigenTrust," in *Proceedings of the 2nd Workshop on Economics of Peer-to-Peer Systems*, (Harvard University, USA), June 2004.

[56] P. Michiardi and R. Molva, "CORE: A collaborative Reputation mechanism to enforce node cooperation in Mobile Ad-hoc networks," in *Proceedings of IFIP TC6/TC11 6th Communications and Multimedia Security* (B. Jerman-Blazic and T. Klobucar, eds.), IFIP, (Portoroz, Slovenia), pp. 107–121, Kluwer, September 2002.

[57] S. Buchegger and J.-Y. L. Boudec, "A robust reputation system for P2P and Mobile Ad-hoc networks," in *Proceedings of the 2nd Workshop on Economics of Peer-to-Peer Systems*, (Harvard University, USA), June 2004.

[58] O. Ardaiz, P. Artigas, T. Eymann, F. Freitag, R. Messeguer, L. Navarro, and M. Reinicke, "Exploring the catallactic coordination approach for peer-to-peer systems," in *Proceedings of Euro-Par* (H. Kosch, L. Böszörményi, and H. Hellwagner, eds.), vol. 2790 of *LNCS*, (Klagenfurt, Austria), pp. 1265–1272, Springer, August 2003.

[59] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent Peer-to-peer storage utility," in *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP)*, (Chateau Lake Louise, Banff, Alberta, Canada), pp. 188–201, ACM, October 2001.

[60] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP)*, (Chateau Lake Louise, Banff, Alberta, Canada), pp. 202–215, October 2001.

[61] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: A decentralized, Peer-to-peer Web cache," in *Proceedings of the 21st Annual ACM Symposium on Principles of Distributed Computing (PODC)*, (Monterey, California, USA), pp. 213–222, ACM, July 2002.

[62] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scribe: a large-scale and decentralized application-level Multicast infrastructure," *IEEE Journal on Selected Areas in Communications*, vol. 20, pp. 1489–1499, October 2003.

[63] S. Zhuang, I. Stoica, D. Adkins, S. Ratnasamy, S. Shenker, and S. Surana, "Internet Indirection Infrastructure," in *Proceedings of 1st International Workshop on Peer-to-Peer Systems* (P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, eds.), vol. 2429 of *LNCS*, (Cambridge, MA, USA), pp. 191–202, Springer, March 2002.

[64] A. Andrzejak and Z. Xu, "Scalable, efficient range queries for Grid information services," in *Proceedings of the Second International Conference on Peer-to-Peer Computing*, pp. 33–40, Linköping University, Sweden, IEEE Computer Society, September 2002.

[65] S. Shenker, S. Ratnasamy, M. Handley, and R. Karp, "Topologically-aware overlay construction and server selection," in *Proceedings of IEEE INFOCOM*, (New York City, NY), June 2002.

[66] B. Krishnamurthy, J. Wang, and Y. Xie, "Early measurements of a cluster-based architecture for P2P systems," in *Proceedings of the 1st Internet Measurement Workshop*, (San Francisco, CA, USA), pp. 105–109, ACM, November 2001.

[67] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," Tech. Rep. TR-20001-26, University of Chicago, 2001.

[68] I. Foster and M. Ripenau, "Mapping the Gnutella network: Macroscopic properties of large-scale Peer-to-peer systems," in *Proceedings of 1st Internation Workshop on Peer-to-Peer Systems* (P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, eds.), vol. 2429 of *LNCS*, (Cambridge, MA, USA), pp. 85–93, Springer, March 2002.

[69] Gnutella 2. `http://sourceforge.net/projects/gnutella2/`

[70] A. D. Joseph, B. Y. Zhao, Y. Duan, L. Huang, and J. D. Kubiatowicz, "Brocade: Landmark routing on overlay networks," in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)* (P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, eds.), vol. 2429 of *LNCS*, (Cambridge, MA), pp. 34–44, Springer, March 2002.

[71] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, "Topology-aware routing in structure Peer-to-peer overlay network," in *International Workshop on Future Directions in Distributed Computing (FuDiCo)*, (Bertinoro, Italy), June 2002.

[72] K. W. Ross, "Hash-routing for collections of shared Web caches," *IEEE Network Magazine*, vol. 11, pp. 37–44, Nov-Dec 1997.

[73] D. Karger, A. Sherman, A. Berkhemier, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi, "Web caching with Consistent Hashing," in *Eighth International World Wide Web Conference*, May 1999.

[74] F. E. Bustamante and Y. Qiao, "Friendships that last: Peer lifetime and its role in P2P protocols," in *Proceedings of the 8th International Workshop on Web Content Caching and Distribution*, (Hawthorne, NY, USA), September 2003.

[75] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," *Operating Systems Review*, vol. 22, pp. 8–32, January 1988.

[76] P. Eugster, S. Handurukande, R. Guerraoui, A. Kermarrec, and P. Kouznetsov, "Lightweight probabilistic broadcast," in *Proceedings of The International Conference on Dependable Systems and Networks (DSN 2001)*, pp. 443–452, 2001.

[77] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet, 3rd edition*. Addison Wesley, May 2003.

[78] VTHD Project. `http://www.vthd.org`.

[79] R. Kilchenman, "Design and implementation of a distributed object storage system on peer nodes," Master's thesis, Universität Mannheim/Institut Eurécom, Sophia-Antipolis, France, 2002.

[80] F. Garcia, "Hierarchical peer-to-peer look-up service prototype implementation," Master's thesis, Public University of Navarra/Institut Eurecom, Sophia-Antipolis, France, 2003.

[81] M. Waldvogel and R. Rinaldi, "Efficient topology-aware overlay network," in *Proceedings of HotNets-I*, October 2002.

[82] E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *Proceedings of INFOCOM*, (New York, USA), IEEE, June 2002.

[83] H. Tangmunarunkit, R. Govindan, S. Shenker, and D. Estrin, "The impact of routing policy on Internet paths," in *Proceeding of INFOCOM*, (Anchorage, Alaska, USA), pp. 736–742, IEEE, 2001.

[84] M. J. Freedman and D. Mazieres, "Sloppy hashing and self-organizing clusters," in *Proceedings of 2nd International Workshop on Peer-to-Peer Systems (IPTPS)* (M. F. Kaashoek and I. Stoica, eds.), vol. 2735 of *LNCS*, (Berkeley, CA, USA), pp. 45–55, Springer, February 2003.

[85] B. Krisnamurthy and J. Wang, "On network-aware clustering of Web sites," in *Proceedings of SIGCOMM*, (Stockholm, Sweden), ACM, August 2000.

[86] J. Wang, *Network Aware Client Clustering and Applications*. PhD thesis, Cornell University, May 2001.

[87] I. Abraham, D. Malkhi, and O. Dobzinski, "Land: Locality aware networks for distributed hash tables," Tech. Rep. TR-2003-75, The Hebrew University of Jerusalem, Israel, June 2003.

[88] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT: A Geographic Hash Table for data-centric storage," in *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications* (C. S. Raghavendra and K. M. Sivalingam, eds.), (Atlanta, Georgia, USA), pp. 78–87, ACM, September 2002.

[89] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-peer informatic system based on the XOR metric," in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)* (P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, eds.), vol. 2429, (Cambridge, MA, USA), pp. 53–65, Springer, March 2002.

[90] M. Waldvogel, *Fast Longest Prefix Matching: Algorithms, Analysis, and Applications*. Aachen, Germany: Shaker Verlag, April 2000.

[91] J. Kangasharju and K. W. Ross, "Adaptive replication and replacement strategies for P2P caching," tech. rep., Institut Euécom, July 2002.

[92] A. Chankhunthod *et al.*, "A hierarchical Internet object cache," in *Proceedings of USENIX Technical Conference*, (San Diego, CA, USA), pp. 153–164, USENIX Association, January 1996.

[93] Oregon University, ""Route Views" archive." `http://rv-archive.uoregon.edu/`

[94] Merit Network, "Internet performance measurement and analysis (IPMA) project." `http://www.merit.edu/~ipma/routing_table/`

[95] Castify Networks. http://www.castify.net/

[96] RIPE. http://www.ripe.net/

[97] NLANR, "BGP-system usage of 32 bit internet address space." http://moat.nlanr.net/IPaddrocc/

[98] "Traceroute site." http://www.traceroute.org/

[99] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts.," in *Proceedings of 2nd Internet Measurement Workshop*, (Marseille, France), November 2002.

[100] L. A. Adamic, "Zipf, Power-laws, and Pareto – a ranking tutorial," tech. rep., Xerox, 2000.

[101] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the IP Multicast service and architecture," *IEEE Network magazine special issue on Multicasting*, vol. 14, pp. 78–88, January/February 2000.

[102] S. Shi and M. Waldvogel, "A rate-based end-to-end multicast congestion control protocol," in *Proceedings of ISCC*, (Antibes, France), pp. 678–686, July 2000.

[103] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proceedings of SIGCOMM*, (Pittsburgh, PA, USA), ACM Press, August 2002.

[104] Y.-H. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A case for end system Multicast," *IEEE Journal on Selected Areas in Communication, Special Issue on Networking Support for Multicast*, vol. 20, October 2002.

[105] S. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide area data dissemination," in *Proceedings of the 11th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, vol. 2429, (Port Jefferson, NY, USA), pp. 11–20, ACM, June 2001.

[106] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker, "Application-level Multicast using content-addressable networks," in *Proceedings of the 3rd Third International COST264 Workshop on Networked Group Communication* (J. Crowcroft and M. Hofmann, eds.), vol. 2233 of *LNCS*, pp. 14–29, Springer, November 2001.

[107] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, "An evaluation of scalable application-level Multicast built using Peer-to-peer overlays," in *Proceedings of INFOCOM*, (San Francisco, USA), IEEE, April 2003.

[108] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An application level multicast infrastructure," in *Proceedings of the 3rd USNIX Symposium on Internet Technologies and Systems (USITS '01)*, (San Francisco, CA, USA), pp. 49–60, Mar. 2001.

[109] P. Francis, "Yoid: Extending the multicast internet architecture," 1999. White paper, http://www.icir.org/yoid/.

[110] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth Multicast in a cooperative environment," in *Proceedings of of the 20th ACM Symposium on Operating System Principle (SOSP)*, LNCS, (New York, USA), pp. 292–303, Springer, October 2003.

[111] J. W. Byersa, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," in *Proceedings of SIGCOMM*, pp. 47–60, ACM Press, August 2002.

[112] Y. Zhu, J. Guo, and B. Li, "oEvolve: Towards evolutionary overlay topologies for high bandwidth data dissemination," *IEEE Journal on Selected Areas in Communications, Special Issue on Quality of Service Delivery in Variable Topology Networks*, Third Quarter 2004.

[113] C. Tang and P. K. McKinley, "A distributed approach to topology-aware overlay path monitoring," in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS)*, (Tokyo, Japan), pp. 122–131, IEEE Computer Society, March 2004.

[114] M. Kwon and S. Fahmy, "Topology-aware overlay networks for group communication," in *Proceedings of NOSSDAV*, (Miami Beach, Florida, USA), pp. 127–136, ACM, May 2002.

[115] A. Riabov, Z. Liu, and L. Zhang, "Overlay Multicast trees of minimal delay," in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS)*, (Tokyo, Japan), pp. 654–664, IEEE Computer Society, March 2004.

[116] D. Kostic, A. Rodriguez, J. Albrecht, A. Bhirud, and A. M. Vahdat, "Using random subsets to build scalable network services," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, (Seattle, WA, USA), March 2003.

[117] A. Rodriguez, D. Kostic, and A. Vahdat, "Scalability in adaptive multi-metric overlays," in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS)*, (Tokyo, Japan), pp. 112–121, IEEE Computer Society, March 2004.

[118] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High bandwidth data dissemination using an overlay mesh," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-19)*, (Bolton Landing, NY, USA), pp. 282–297, October 2003.

[119] P. Rodriguez, A. Kirpal, and E. W. Biersack, "Parallel-access for mirror sites in the internet," in *Proceedings of INFOCOM*, (Tel-Aviv, Israel), pp. 864–873, IEEE, March 2000.

[120] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. Al Hamra, and L. Garcés-Erice, "Dissecting bittorrent: Five months in a torrent's lifetime," in *Proceedings of Passive*

and Active Measurements (C. Barakat and I. Pratt, eds.), vol. 3015 of *LNCS*, (Juan-les-Pins, France), pp. 1–11, Springer, April 2004.

[121] D. Tran, K. Hua, and T. Do, "ZIGZAG: An efficient peer-to-peer scheme for media streaming," in *Proceedings of IEEE Infocom*, (San Francisco, USA), IEEE, March 2003.

[122] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan, "Detour: A case for informed internet routing and transport," *IEEE Micro*, vol. 19, Jan. 1999.

[123] M. Pias, J. Crowcroft, S. Wilbur, and T. H. S. Bhatti, "Lighthouses for scalable distributed location," in *Proceedings of 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, vol. 2735 of *LNCS*, (Berkeley, CA, USA), pp. 278–291, Springer, February 2003.

[124] H. Lim, J. C. Hou, and C.-H. Choi, "Constructing Internet coordinate system based on delay measurement," in *Proceedings of the 3rd Internet Measurement Conference (IMC)*, (Miami Beach, FL, USA), pp. 129–142, October 2003.

[125] L. Tang and M. Crovella, "Virtual landmarks for the internet," in *Proceedings of the Internet Measurement Conference (IMC)*, (Miami Beach, Florida, USA), ACM, October 2003.

[126] CAIDA. `http://www.caida.org/`

[127] S. van Langen, X. Zhou, and P. V. Mieghem, "On the estimation of Internet disctances using landmarks," in *Proceedings of the International Conference on Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN)*, (St. Petersburg, Russia), February 2004.

[128] M. Balazinska, H. Balakrishnan, and D. Karger, "INS/Twine: A scalable Peer-to-peer architecture for intentional resource discovery," in *Proceedings of the International Conference on Pervasive Computing* (F. Mattern and M. Naghshineh, eds.), vol. 2414 of *LNCS*, (Zürich, Switzerland), pp. 195–210, Springer, August 2002.

[129] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The design and implementation of an intentional naming system," in *Proceedings of the 17th ACM Symposium on Operating System Principles (SOSP)*, (Charleston, South Carolina, USA), pp. 186–201, ACM Press, December 1999.

[130] M. Harren, J. Hellerstein, R. Huebsch, B. Loo, S. Shenker, and I. Stoica, "Complex queries in DHT-based Peer-to-peer networks," in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)* (P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, eds.), vol. 2429 of *LNCS*, (Cambridge, MA, USA), pp. 242–259, Springer, March 2002.

[131] A. Gupta, D. Agrawal, and A. Abbadi, "Approximate range selection queries in Peer-to-peer systems," Tech. Rep. UCSB/CSD-2002-23, University of California at Santa Barbara, 2002.

[132] O. Sahin, A. Gupta, D. Agrawal, and A. Abbadi, "Query processing over Peer-to-peer data sharing systems," Tech. Rep. UCSB/CSD-2002-28, University of California at Santa Barbara, 2002.

[133] J. Mischke and B. Stiller, "Rich and scalable peer-to-peer search with shark," in *Proceedings of the 5th Annual International Workshop on Active Middleware Services (AMS'03)*, (Seattle, WA, USA), pp. 112–121, IEEE, June 2003.

[134] J. Li, B. T. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morris, "On the feasibility of Peer-to-peer Web indexing and search," in *Prooceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, vol. 2735 of *LNCS*, (Berkeley, CA, USA), pp. 207–215, Springer, February 2003.

[135] C. Tang, Z. Xu, and M. Mahalingam, "pSearch: Information retrieval in structured overlays," in *Proceedings of ACM HotNets-I*, Oct. 2002.

[136] P. Reynolds and A. Vahdat, "Efficient peer-to-peer keyword searching," in *Proceedings of the ACM/IFIP/USENIX International Middleware Conference*, vol. 2672 of *LNCS*, (Rio de Janeiro, Brazil), pp. 21–40, Springer, June 2003.

[137] D. Bauer, P. Hurley, R. Pletka, and M. Waldvogel, "Bringing efficient advanced queries to distributed hash tables," in *Proceedings of IEEE LCN*, (Tampa, Florida, U.S.A.), Nov. 2004.

[138] I. Foster, A. Iamnitchi, and M. Ripeanu, "Locating data in (small-world?) Peer-to-peer scientific collaborations," in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)* (P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, eds.), vol. 2429 of *LNCS*, (Cambridge, MA, USA), pp. 85–93, Springer, March 2002.

[139] W3C, "Extensible Markup Language (XML) 1.0 (3rd Edition)." `http://www.w3.org/TR/REC-xml/`, February 2004.

[140] D. Connolly, *XML – Principles, Tools, and Techniques*. O'Reilly, October 1997.

[141] DBLP, "Bibliography search engine." `http://dblp.uni-trier.de/`

[142] W3C, "XML Path Language (XPath) 1.0." `http://www.w3.org/TR/xpath/`, November 1999.

[143] BibFinder, "Bibliography search engine." `http://kilimanjaro.eas.asu.edu/`

[144] NetBib, "Bibliography search engine." `http://edas.info/S.cgi?search=1`.

[145] CiteSeer, "Bibliography search engine." `http://citeseer.nj.nec.com/cs/`

[146] G. Zipf, *Human Behavior and the principle of least effort*. Addison-Wesley Press, 1949.

[147] Google, "Internet search engine." `http://www.google.com/`

[148] N. Alon, M. Dietzfelbinger, P. B. Miltersen, E. Petrank, and G. Tardos, "Linear hashing," *Journal of the ACM*, vol. 46, no. 5, pp. 667–683, 1999.

[149] CDDB, "CD identification Database." http://www.cddb.org/

[150] Y. Shavitt and T. Tankel, "Big-bang simulation for embedding network distances in Euclidean space," in *Proceedings of INFOCOM*, (San Francisco, CA, USA), IEEE, April 2003.

[151] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris, "Practical, distributed network coordinates," in *Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II)*, (Cambridge, Massachusetts), ACM SIGCOMM, Nov. 2003.

[152] NetGeo, "Geographic location service – CAIDA." http://netgeo.caida.org/perl/netgeo.cgi.